LA BIBLIA DEL





DISTRIBUYE:

R. B. A. PROMOTORA DE EDICIONES, S. A. - Polígono Industrial Zona Franca - Sector B, calle B, núm. 11 - 08004-Barcelona



La biblia del AMSTRAD

Cristian Longhi



Grupo de Trabajo Software, S. A.
Bailen, 20, 1.° Izq. 28005 MADRID
Depósito Legal: M. 1.423-1986
Imprime: Gráf. FUTURA, Sdad. Coop. Ltda.



La biblia del AMSTRAD



INTRODUCCION

El lenguaje de BASIC fue desarrollado en los años sesenta, en New Hampshire, Estados Unidos, por los profesores John Kemeny y Thomas Kurt, para aprender de un modo sencillo su uso, así como una implementación muy económica.

Estos dos profesores lo escribieron de tal modo, que muchos usuarios pudieran usar un solo Computador para fines de aprendizaje, siendo un lenguaje interactivo, es decir un lenguaje en el que el usuario obtenga una respuesta inmediata a lo que se introduce por el teclado de la Computadora.

Desde su implementación, el lenguaje BASIC ha experimentado una gran evolución, hasta el punto de que ha llegado a ser el lenguaje estandar de las microcomputadoras, siendo su resultado de gran ayuda para los usuarios que no son principiantes, utilizándose, además de entretenimiento y enseñanza, para actividades comerciales y aplicaciones industriales.

La BIBLIA DEL AMSTRAD es una obra destinada al BASIC del AMSTRAD, así como todo lo relacionado con este Computador, para facilitarle paso a paso el aprendizaje de este lenguaje y así poder desarrollar sus propios programas convirtiéndose en un programador avanzado.



Como hemos mencionado anteriormente, una vez que Vd. termine de leerse esta obra, se sentirá más animado para sacar el mejor fruto a su Computador, dado que lo peor que le puede a uno pasar es tener este tipo de máquinas delante de uno y no saber qué hacer con ellas; por lo tanto, le recomendamos que siga paso a paso a todos los capítulos sin saltarse ninguno, pues de lo contrario no podrá completar su educación en el campo de las Computadoras.

Esta obra consta de treinta y cinco Capítulos más tres Apéndices, donde Vd. podrá aprender el uso de su Computador desarrollando los ejercicios que hay en muchos de los mismos, y comprobando su resultado con los que aparecen en el Apéndice 1.

Además de todo esto, hay un Apéndice con programas donde Vd. podrá introducirlos en su Computador, salvándolo en cinta o disco, para su utilización y así poderlos modificar para experimentar con el lenguaje de BASIC.

Quiero expresar mi agradecimiento a la compañía HISPANOMICRO, S.A., por las facilidades que me han dado en desarrollar esta obra, así como en la comprobación de la misma.

Cristian Longhi
Autor



CAPITULO 1

INTRODUCCION AL COMPUTADOR

En el momento en que Vd. conecta su Computador AMSTRAD, tendrá que seguir unas reglas determinadas para poderle sacar un buen partido a su máquina. Estas reglas están permanentemente en el computador en dos programas, llamados, uno Monitor y el otro el Interpretador.

Empezaremos en este Capítulo dialogando con el computador, enseñándole las etiquetas más fundamentales del mismo y creando su primer programa.

Una vez conectada la máquina a la red y procediendo a encenderla, después de unos segundos de calentamiento del tubo del monitor (Color o Fósforo Verde), aparecerá en la parte superior de la pantalla, el siguiente mensaje:

```
Amstrad 64K Microcomputer (V1)
© 1984 Amstrad Consumer Electronics plc
           and Locomotive Software Ltd

BASIC 1.0
Ready
█
```



Si se presiona la tecla de **ENTER**, aparecerá otra vez en los renglones siguientes.

```
█  (cursor)
```

Para estar seguro de empezar con una memoria limpia, tendrá que teclear la palabra **NEW** y luego dar a la tecla **ENTER**, y el computador le responderá en el renglón siguiente con:

```
Ready
█
```

Si desea limpiar totalmente la pantalla, deberá teclear la palabra **CLS** y luego dar a la tecla **ENTER**, apareciendo en la parte superior de la pantalla:

```
Ready
█
```

Otro de los pasos de gran utilidad, es el inspeccionar el estado de la memoria (ver qué cantidad de memoria se ha usado), para esto habrá que teclear la palabra **PRINT MEM** y luego dar a la tecla **ENTER.**

```
Ready
█
PRINT MEM
```

0 (si no se ha usado la memoria)

```
Ready
█
```

¿QUE ES UN PROGRAMA DE COMPUTADORA?

Un programa es una secuencia de instrucciones que el computador almacena hasta que se le da la orden de ejecutarlo. Muchos de los programas que se usan en el Computador AMSTRAD están escritos en el lenguaje BASIC, con lo cual este libro enfocará los diversos comandos de este tipo de lenguaje.

Empezaremos escribiendo un programa de una sola lí-



nea para que su computador se presente. Primero nos aseguraremos de que su memoria está limpia, tecleando la palabra **NEW** y luego dar a la tecla **ENTER** para que aparezca la palabra **Ready**.

A continuación teclee la siguiente línea:

```
10 PRINT "Buenas .. soy su Computador AMSTRAD!!"
```

No de a la tecla **ENTER** por ahora.

Todos los comandos los podrá escribir en minúsculas si lo desea, como en este caso **print**. Si se desea usar letras mayúsculas en el texto, habrá que oprimir la tecla **SHIFT**, aunque Vd. podrá comprobar en su teclado que hay teclas con doble tipo de caracteres, así que también deberá presionar la tecla **SHIFT** si tiene que usar los caracteres marcados en la parte superior de la misma.

Si al teclear cometiese algún error, no se preocupe, dado que usando la tecla **DEL** corregirá la equivocación, cosa más sencilla que cuando se usa una máquina de escribir. Para más información en el manejo de esta tecla de borrado, ver el manual de su AMSTRAD.

Ahora volveremos a examinar detalladamente lo que Vd. ha escrito en su computador.

1.— ¿Ha puesto entre comillas la frase que va después de **PRINT**?.

2.— ¿Hay comillas de más en la frase escrita?

Si todo está bien podrá pulsar la tecla **ENTER** y aparecerá el cuadradito del cursor. Si por casualidad hay un error en la línea después de haber pulsado la tecla **ENTER**, la tecla **DEL** no tendrá ningún efecto, debiendo entrar otra vez en la línea completa que reemplazará a la anterior una vez p la tecla **ENTER**. Esto es debido a que ambas comparten el mismo número de línea (en este caso el 10) y la última línea entrada es la válida para el computa-



dor. En otros Capítulos enseñaremos cómo editar una línea sin tener que teclearla completamente.

En el segundo paso, vamos a ejecutar nuestro pequeño programa, y para eso usaremos un comando de Basic llamado **RUN**. Así pues, teclearemos:

```
RUN
```

y después **ENTER**

Si no hay ninguna equivocación aparecerá en la pantalla:

```
Buenas .. soy su Computador AMSTRAD!!
Ready
█
```

En el caso de que no funcionase, teclee otra vez la palabra **RUN**, y si sigue sin funcionar, entre la palabra **NEW**, y otra vez la línea completa 10.

Una vez que este pequeño programa funcione comprobará que la palabra **PRINT** y las comillas no aparecen en la pantalla, dado que son parte de las instrucciones del programa.

Entre otra vez la palabra **RUN** y accione la tecla **ENTER** y verá que se repite la misma respuesta en pantalla una y otra vez dado que este programa permanece en memoria hasta que Vd. lo borra con el comando **NEW** o desconectando el computador. La computadora es informada de ejecutar un programa o un comando cuando se acciona la tecla **ENTER**, por eso si introduce una línea o un comando, siempre deberá usar dicha tecla para que se ejecute.



CAPITULO 2

DESARROLLANDO UN PROGRAMA

En este momento tenemos un programa en la memoria del computador (si Vd. ha desconectado el computador entre estos dos Capítulos, vuélvalo a conectar y teclee la línea 10 de Capítulo 1). Este programa consta de una sola línea, pero lo vamos a ampliar añadiendo una segunda línea.

En Basic cada línea del programa debe llevar un número y el programa se ejecuta en un orden ascendente, desde la línea con una numeración más baja, hasta la que tiene la numeración más alta.

Nota: Cuando aparezca la palabra en la forma de ENTER a través de todo este libro, significa que debe pulsar la tecla ENTER.

Entre la siguiente línea desde su teclado:

```
20 PRINT "Desea Vd. mis servicios ..?"
ENTER
```

Compruebe si todo está correcto, especialmente las comillas, y prosiga con el siguiente paso.




```
RUN
ENTER
```

Si todo está correcto, aparecerá en la pantalla lo siguiente:

```
Buenas .. soy su Computador AMSTRAD!!
Desea VD. mis servicios ..?
Ready
```

Si Vd. responde a la pregunta con un:

```
SI
ENTER
```

Le aparecerá en la pantalla un mensaje de error, como **Syntax error**.

Deliberadamente le hemos forzado a crear un error, para poder demostrar la capacidad de detección de errores de su Computador, el cual es lo suficientemente inteligente para indicarle cuándo Vd. ha cometido un error y la naturaleza del mismo.

En este caso, el Computador está esperando una nueva línea de programa o un comando de Basic, más adelante le demostraremos cómo haremos que acepte una respuesta de «SI» o «NO».

Hay docenas de posibles errores que se pueden hacer, y más adelante le enseñaremos a crear un «CODIGO DE ERRORES», pero mientras tanto sólo tocaremos una situación de error la cual la podremos reconocer dado que se refiere a error de transcripción.

Volveremos a teclear la línea 20 para crear un error, así pues la pondremos de la siguiente forma:

```
20 PRIMT "Desea Vd. mis servicios ..?"
ENTER
RUN
ENTER
```



Otra vez obtendremos un mensaje de ERROR:

```
Syntax Error in
20 PRIMT "Desea Vd. mis servicios ..?"
```

Deberá pulsar la tecla **ENTER** y escribir de nuevo la línea 20 como se indica al principio de este Capítulo. Más adelante le indicaremos otra forma de poder corregir esta línea antes de pulsar la tecla **ENTER**.

Es por norma el espaciar la numeración de las líneas de 10 en 10 para facilitar la inserción de otras líneas si es necesario para el desarrollo del programa o la modificación del mismo, así pues introduciremos una línea entre la 10 y la 20 para ver lo que pasa:

```
15 PRINT
ENTER
RUN
ENTER
```

En la pantalla deberá aparecer:

```
Buenas .. soy su Computador AMSTRAD!!
Desea Vd. mis servicios ..?
```

El resultado aparecerá con un espacio entre las dos líneas que hacen que el programa sea más legible. Cada comando **PRINT** que Vd. inserte, sin ningún mensaje, le dará un salto de línea en el resultado.

Otra importante sentencia es **REM**, REMARK (ANOTACION), y significa que Vd. puede introducir en su programa cualquier número de **REM** para facilitarle cualquier anotación en el mismo, como por ejemplo, el tipo de programa, qué hace este programa, o cómo se ejecuta el programa, es como tener un cuaderno de anotaciones dentro de su programa.

Cuando le indica a su Computador que ejecute su programa, donde se encuentran diferentes **REM**, éste saltará



las líneas donde se encuentren estas sentencias, dado que no tienen ningún efecto en su ejecución.

Insertar la siguiente línea:

```
5 REM * Este es mi primer programa *
ENTER
RUN
ENTER
```

La ejecución del programa dará un resultado como el anterior, dado que la línea 5 no afecta al mismo.

Dado que el programa está tomando unas dimensiones aceptables, y para ver lo que llevamos escrito, podremos hacer uso de otro comando de Basic llamado **LIST**, así pues teclearemos:

```
LIST
ENTER
```

Y obtendremos lo siguiente en la pantalla:

```
5 REM * Este es mi primer programa *
10 PRINT "Buenas .. soy su Computador AMSTRAD!!"
15 PRINT
20 PRINT "Desea Vd. mis servicios ..?"
Ready
█
```

Vd. podrá acceder al comando **LIST** en cualquier momento en que el cuadradillo del cursor aparezca a la izquierda de la pantalla.

¿Como finalizar un programa?

Para finalizar un programa bastará con la ejecución de la última línea del mismo, pero muchos Computadores requieren el uso de la sentencia **END** así el computador sabe que ha terminado, pero en su Computador AMSTRAD esta sentencia es opcional pudiéndola poner o dejando de ponerla.



Analizando la sentencia **END** veremos, que por norma se la da un número de línea muy alto, por ejemplo 99 o 999 o 9999, dependiendo del último número de línea de su programa y limitado por la numeración que el computador podrá aceptar, que en este caso es de 65529. Así pues introduciremos la sentencia **END** en nuestro programa poniendo lo siguiente:

```
99 END
ENTER
RUN
ENTER
```

El ejemplo una vez ejecutado saldrá en pantalla de la siguiente forma:

```
Buenas .. soy su Computador AMSTRAD!!

Desea Vd. mis servicios ..?
Ready
█
```

¿Por qué razón no ha aparecido en la pantalla la palabra **END**?, es muy sencillo, dado que sólo sale en pantalla el «objeto» de la sentencia **PRINT.**

Borrar una línea sin ser reemplazada

Como ejercicio le mostraremos cómo eliminar una línea del programa sin ser reemplazada. Tomemos por ejemplo la línea 99 que queremos pasarla a la línea 65529, esto requerirá dos pasos, primero eliminaremos la línea 99 escribiendo lo siguiente:

```
99
ENTER
```

En este momento la línea 99 ha sido borrada del programa, y para estar seguros listaremos el mismo entrando:

```
LIST
ENTER
```

En la pantalla sólo aparecerán las líneas 5, 10, 15 y 20,



así de esta manera también se podrá borrar cualquier línea. El segundo paso será el insertar la nueva línea, por lo que introduciremos lo siguiente:

```
65529 END
ENTER
```

Listaremos otra vez el programa con el comando **LIST** y veremos que ahora tenemos las líneas 5, 10, 15, 20 y 65529.



CAPITULO 3

EL EDITOR

El Editor es de un valor extraordinario dentro del Basic y su misión es la de editar un programa. Es un comando facilísimo de usar y a su vez muy útil a la hora de necesitarlo dado su gran potencia. El Editor se verá a través de este libro en varios capítulos, pero en éste les explicaremos sus diferentes funciones.

Limpie la memoria y pantalla usando **NEW** y **CLS** para así no confundirse con lo que tenga en ella.

El cursor es controlado con las cuatro teclas con flechas situadas a la derecha superior de su teclado con una tecla en el centro marcada con la palabra **COPY**.

Para demostrar las diferentes funciones de corrección con el editor, entraremos una línea numérica con errores:

10 FOR A = 100 TO 50 (El error está en el 100 que debería ser 10)

Para corregir tenemos dos opciones:

1.— Se puede pulsar la tecla **ENTER** y teclear toda la línea, la cual sustituirá a la anterior.



2.— Situar al cuadradillo con vídeo inverso encima del segundo cero de la cifra 100 y pulsar la tecla **DEL** para que quite ese cero lo que hará que el cuadradillo se mueva hacia la izquierda. Otra manera es el situar el cuadradillo encima de cualquiera de los dos ceros y pulsar la tecla **CLR** desapareciendo el carácter en que se encuentre dicho cuadradillo. Pulsando la tecla **ENTER**, la línea corregida sustituirá a la que tiene el error quedando el programa corregido.

Este método se usa cuando se ha citado una línea y se ha detectado el error antes de pulsar la tecla **ENTER**, o si al correr el programa le da un **Syntax error** con lo cual verá que el cuadradillo está situado encima del primer número de la línea.

Si el Computador detecta un error y no te muestra la línea completa con dicho error, sino que solamente te indica el número de línea, deberás listar dicha línea con el comando **LIST n** (donde n es el número de la línea). Una vez listada dicha línea y detectado el error, Vd. podráa corregirlo usando las teclas **SHIFT** y (↑) para situar el cuadradillo encima del número de línea y luego con la tecla (→) situarlo encima del carácter a corregir. Una vez que se efectúe dicha corrección, se pulsará la tecla **ENTER** para introducir dicha línea en memoria.

Otra forma de hacer correcciones en un programa, es el usar la tecla **COPY** (copiar) en unión con lo anteriormente expuesto. Para esto deberá listar la parte del programa donde Vd. cree que está el error con el comando **LIST n1-n2** (donde n1 es el número de línea más pequeño y n2 es el número más grande del programa), y con las teclas **SHIFT** y (↑) simultáneamente colocar el cuadradillo encima del carácter a corregir borrándolo e introduciendo el correcto; efectuada esta corrección, pulsar la tecla **ENTER** para meter la línea en memoria. Otra ventaja de este procedimiento es el poder copiar líneas de texto completas cuando se necesita duplicidad de ellas, para esto el procedi-



miento es muy similar al anterior. primero listar la línea que queremos duplicar y luego usando las teclas de **SHIFT** y (↑) simultáneamente y con la tecla (→) situar el cuadradillo encima del primera carácter, escribiremos el nuevo número de línea y con la tecla **COPY** copiaremos el texto entero. Una vez esté confeccionada la nueva línea pulsaremos la tecla **ENTER** para introducir en memoria dicha línea.

Supongamos que tenemos un programa como el que se muestra a continuación:

```
5 CLS
10 FOR A = 0 TO 255
20 PRINT CHR$(A);
30 NEXT A
```

Puedes pasar a la forma de **EDIT** (Edición) tecleando:

```
EDIT 20
```

El Computador mostrará la línea 20 de la siguiente forma:

20 PRINT CHR$(A); (con el cuadradillo encima del 2)

A continuación Vd. puede usar las teclas del cursor conjuntamente con las de **CLR** y **DEL** tal como se ha indicado anteriormente para corregir, y una vez comprobada dicha línea pulsar la tecla **ENTER**.

Si en el transcurso de la edición o corrección de errores, se complica la cosa y no sabe que hacer, con sólo pulsar la tecla **ESC** se saldrá de esa rutina sin afectar nada a la línea o texto que se esté corrigiendo o editando.



CAPITULO 4

NUMERACION AUTOMATICA DE LINEA (AUTO)

El comando **AUTO** es uno de los más cómodos para editar en Basic dado que te numera las líneas y te da el espacio correspondiente entre el número y el texto.

Para la práctica de este comando, empezaremos borrando la memoria entrando NEW y si se quiere borrar la pantalla con el **CLS**.

Desde el teclado entre la siguiente palabra:

```
AUTO
```

La pantalla le responderá con:

```
10 ▒
```

En la posición donde el cursor está, teclee lo siguiente:

```
PRINT "Que estamos haciendo aqui"
ENTER
```

En la pantalla aparecerá la siguiente línea:

```
20 ▒
```

Teclee lo siguiente donde está posicionado el cursor:




```
PRINT "Esto es ridiculo"
ENTER
```

Y aparecerá otra vez:

```
30 █
```

Todo esto se asemeja mucho al EDITOR, pero no lo es, sino que es un modo de numeración Automática de Líneas.

Es evidente que en este punto estamos introduciendo números de líneas tantas veces como presionemos la tecla **ENTER**, y la única forma de salirse del comando AUTO es presionando la tecla **ESC**.

Entre el comando **LIST** y verá que solamente le listará las líneas que contienen datos, que en este caso son las 10 y 20. Borre la memoria tecleando la palabra **NEW** y entre lo siguiente:

```
AUTO 1000,200
```

Presione unas seis veces la tecla de **ENTER** y verá que en la primera línea aparecerá un 1000 y en la segunda 1200 y así sucesivamente en incrementos de 200. Esto quiere decir que en el comando **AUTO 1000,200;** el 1000 indica el comienzo de la numeración de la primera línea, y el 200 el incremento entre líneas. Por ejemplo, si nosotros entramos **AUTO 100,5** la primera línea empezará con la numeración 100 e incrementará en 5.

Otra de las indicaciones del comando **AUTO** es el que aparece un asterisco después del número de línea, por ejemplo **10***. Esto indica que en esa línea hay datos por lo que si pulsamos la tecla **ENTER** borraremos completamente la línea 10, así que si deseamos salvarla se deberá pulsar la tecla **ESC** para salirse del comando **AUTO**.



CAPITULO 5

OPERADORES MATEMATICOS

El lenguaje de Basic usa las cuatro operaciones fundamentales de matemáticas, más una quinta que es una modificación de dos de las otras.

1.— Suma, que usa el símbolo +
2.— Resta, que usa el símbolo –
3.— Multiplicación, que usa el símbolo *
4.— División, que usa el símbolo /
5.— Negación (que significa, multiplicar tanta veces menos una), y usa el símbolo –

Hay que tener cuidado para no usar una x como símbolo de multiplicación, dado que solamente reconocen los Computadores el símbolo * para este tipo de función.

Así pues empezaremos a practicar con las operaciones matemáticas, y para esto borraremos la memoria y la pantalla con los comandos **NEW** y **CLS,** tecleando también el comando **LIST** con lo que dado que la memoria está limpia aparecerá en nuestra pantalla:

```
Ready
█
```



Ahora usaremos el Computador para hacer algunos problemas sencillos, usando ecuaciones, pero para que no se preocupe le diré que las ecuaciones son pequeñas sentencias. Empezaremos con la clásica ecuación de distancia recorrida es igual a velocidad hora por el tiempo utilizado en el viaje.

Para reducir la ecuación usaremos letras llamadas variables que reemplazarán a las tres cantidades. Por ejemplo si la distancia la representamos con una **D**, la velocidad con una **V**, y el tiempo con una **T**, tendremos lo siguientes:

```
40 D = V * T
```

Vd. se preguntará qué hace el 40 aquí, pues es el número de líneas del programa dado que cada paso del programa tiene que tener uno. La razón por la que ponemos el número 40, es dado que hay que dejar espacio libre y que la ecuación sea fácil de leer. Más adelante cuando Vd. escriba programas más largos deseará eliminar espacios libres dado que ocupan espacio de memoria, pero por ahora lo dejaremos así.

Para la designación de las variables, se pueden usar cualquiera de las 26 letras, de la A a la Z, pero siempre es aconsejable el usar letras que se identifiquen con lo que quieren representar, como en este caso D, V, y T para Distancia, Velocidad, y Tiempo.

Para complicar un poco este sencillo ejemplo, vamos a mencionar que hay una forma opcional para escribir esta ecuación, usando la sentencia de Basic **LET**:

```
40 LET D = V * T
```

El uso de **LET** nos recuerda que D es igual a la multiplicación de V por T en vez de la forma tradicional de 1+1=2.

Continuemos con nuestro programa asumiendo que: la distancia (en KM) es igual a la velocidad (en Km por hora) multiplicado por el tiempo (en horas). Así pues qué distancia hay desde Nueva York a Roma, si la velocidad media



del avión es de 500 km por hora y en el viaje se han empleado 12 horas?

Escribiremos el siguiente programa:

```
10 REM * Problema de Distancia, Velocidad, y Tiempo *   ENTER
20 V = 500    ENTER
30 T = 12     ENTER
40 D = V * T    ENTER
```

Compruebe el programa despacio y ejecútelo con:

```
RUN         ENTER
```

Veremos que la respuesta que nos da el Computador es de:

```
Ready
█
```

No crea que su Computador no funciona, lo que pasa que ha efectuado la operación de multiplicación, pero no le hemos introducido ninguna instrucción para que nos dé el resultado.

EJERCICIO 5-1

Este ejercicio le pide a Vd. que termine este programa para que le dé la respuesta en pantalla de esta multiplicación. Esto le servirá de recordatorio y así empezará a ejercitar sus dotes de programador. (Le daremos unas pistas, dado que en capítulos anteriores hemos usado la sentencia **PRINT** para reproducir en pantalla un mensaje, pues bien, ¿si ponemos **50 PRINT "D"**? **... veremos que lo único que conseguimos es el carácter D**, así pues que pasará si quitamos las comillas a la D:).

NO LEA MAS DE ESTE CAPITULO HASTA QUE CONSIGA SOLUCIONAR ESTE EJERCICIO.

Muy bien,la respuesta de 6000 es correcta, pero la presentación no es muy ortodoxa y se parece más a la de una calculadora, así pues volveremos a la sentencia **PRINT** para ver si lo podemos componer.



Vea que en la línea 50 hemos puesto **50 PRINT D** sin comillas, la razón es muy sencilla, hemos querido imprimir sólo la variable **D** del programa. Así pues, como esta respuesta aparece sin ninguna información, introduciremos una línea con una representación más amplia, usando también la sentencia **PRINT**:

```
50 PRINT "LA DISTANCIA EN KILOMETROS ES",D
ENTER
RUN
```

El resultado en pantalla será:

```
LA DISTANCIA EN KILOMETROS ES
6000
```

Este resultado es más profesional y comprobará que el mensaje que está entre las comillas, aparece en la pantalla, además de la cifra de la variable **D**. La coma que hemos puesto, hace que esta cifra se desplace a la siguiente línea, dado que la indicación de la coma es de proseguir en la siguiente línea.

Con este concepto en mente, veamos si podemos cambiar la línea 50 para que el Computador dé un resultado como éste:

```
LA DISTANCIA ES 6000 KILOMETROS
```

Para que la pantalla nos dé este resultado, tendremos que dividir el mensaje en dos partes, introduciendo entre la variable el signo de punto y coma y así el computador sabrá que después del primer punto y coma hay una información y después de esta información continúa el mensaje dado que le hemos puesto otro punto y coma después de la variable. Así pues la línea 50 quedará de esta manera:

```
50 PRINT "LA DISTANCIA ES"; D; "KILOMETROS"
```

Hasta ahora hemos cubierto los suficientes comandos, sentencias y operaciones matemáticas como para resolver diferentes problemas, así pues empezaremos a ejercitar



con todo lo que sabemos resolviendo algunos ejercicios de programación.

EJERCICIO 5-2

Escriba un programa para calcular el tiempo que se necesita para viajar en avión desde París a Los Angeles, donde la distancia es de 6000 kilómetros y el avión tiene una velocidad de crucero de 500 kilómetros por Hora.

EJERCICIO 5-3

Si la circunferencia de un círculo se calcula multiplicando su diámetro por PI (3.14), escriba un programa para calcular la circunferencia de un círculo donde su diámetro es de 35 Metros.

EJERCICIO 5-4

Si el área de un círculo se calcula multiplicando el valor de PI (3.14) por el radio de la circunferencia al cuadrado, escribir un programa para calcular el área de un círculo donde su radio mide 5 metros.

EJERCICIO 5-5

Si en su cuenta corriente tiene un saldo de 22500 pesetas, y VD. usa unos cheques por valor de 170, 350, y 2250, y hace dos depósitos de 400 y 20000, escriba un programa para actualizar su balance de la cuenta corriente.



CAPITULO 6

ANOTACIONES CIENTIFICAS

En el mundo de las matemáticas, hay resultados y cifras muy grandes y muy pequeñas como en el orden de milésimas y billones. Para poder trabajar con estas cifras, los Computadores usan lo llamado **Anotación Exponencial**. Por ejemplo, el número 5 millones (5.000.000) se podrá escribir como **5E + 06**, que significa el número 5 seguido de seis ceros, o técnicamente 5*10 a la sexta potencia (5*10*10*10*10*10*10).

Si en una de las soluciones a un problema aparece **5E-06** esto significará que el punto decimal que está después del 5 habrá que desplazarlo seis lugares a la izquierda, o sea 0.000,005, o lo que técnicamente se llama 5 por 10 a la menos sexta potencia.

Esto no resulta muy difícil una vez se practique un poco para acostumbrarse a poder seguir el curso de los lugares decimales y de los ceros a contar. Dado que los Computadores insisten en usar este método y por ahora no hay ninguno que trabaje de otra forma, practicaremos un poco con los dos ejemplos que ponemos a continuación.



Teclee los comandos **NEW** y **CLS** antes de empezar con los ejercicios para no confundirse con lo anterior y tener la memoria limpia.

EJERCICIO 6-1

Si un millón de coches recorrieron cada uno diez mil kilómetros en un año, ¿cuántos kilómetros habrán recorrido todos ellos en ese año? Escriba y ejecute este simple programa y verá el resultado.

EJERCICIO 6-2

Cambie las líneas 20 y 30 del programa del EJERCICIO 6-1 para expresar los números en Anotación Exponencial, y ejecute el programa.



CAPITULO 7

LOS PARENTESIS Y OTRAS OPERACIONES

Los paréntesis juegan un importante papel en la programación donde hay operaciones matemáticas, y son usados de la misma forma pero con unas excepciones:

1.— En Basic, los paréntesis pueden albergar las operaciones que se han de ejecutar, siendo éstas las que se ejecutarían antes que las que están fuera del paréntesis.

2.— Asimismo, las operaciones que están entre paréntesis dentro de otros paréntesis, también se ejecutarán primero.

3.— Cuando hay una serie de operaciones ligadas entre sí, las cuales están entre paréntesis, el Computador ejecutará la línea de izquierda a derecha resolviendo las multiplicaciones y las divisiones, y empezará otra vez a la izquierda para resolver las sumas y las restas.

NOTA: Las funciones **INT, RND y ABS** se ejecutan antes que las multiplicaciones y las divisiones.

4.— Un problema listado como (X) (Y) no será interpretado por el Computador como una multiplicación de X por Y (X*Y).



EJEMPLO

Si Vd. desea convertir la temperatura de grados Fahrenheit a grados Centígrados (Celsius), habrá que usar la siguiente relación:

Los grados Fahrenheit es igual a 36 grados más nueve quintos de grados Centígrados.

O lo que es igual a esta fórmula:

$$F = (9/5) * C + 32$$

Vamos a asumir que tenemos una temperatura de 25 grados Centígrados, y los queremos convertir a grados Fahrenheit. Tendremos que Editar el siguiente programa, por lo tanto limpiar la memoria y la pantalla con **NEW**y **CLS**.

```
10 REM * CONVERSION DE GRADOS CENTIGRADOS A FAHRENHEIT *
20 C = 25
30 F = (9 / 5) * C + 32
40 PRINT C ; "GRADOS CENTRIGRADOS ="; F ;"GRADOS FAHRENHEIT."
RUN
```

Lo que saldría en pantalla una vez ejecutado el programa sería:

```
25 GRADOS CENTIGRADOS = 77 GRADOS FAHRENHEIT.
```

Lo primero que vemos en la línea 40 es una sentencia de PRINT seguida de cuatro expresiones, dos variables y dos grupos de palabras entre comillas llamadas **LITERALES o CADENAS**.

En la línea 30 nos encontramos con la división de 9/5 que se encuentra entre paréntesis, la cual la multiplicamos por la variable C y al resultado le sumamos 32. Ahora quitaremos los paréntesis; en la línea 30 ejecutaremos el programa y veremos con asombro que la respuesta es la misma **POR QUE???**



1.— En la primera pasada, el Computador empieza resolviendo la operación que está entre paréntesis, en este caso (9/5) dando un resultado de 1.8 el cual es multiplicado por el valor de C (25) y finalmente sumando 32.

2.— En la segunda ejecución, la cual ha sido quitando los paréntesis, el Computador simplemente ha empezado a calcular de izquierda a derecha encontrándose primero con la división de 9/5, luego con la multiplicación por el valor C y finalmente con la suma de 32. Dado que solamente se encontraba un solo paréntesis al principio, no había ninguna diferencia entre la primera y la segunda ejecución.

El siguiente paso que probaremos será el poner el +32 delante de la ecuación, o sea 32 + 9/5 * C, pero sin paréntesis y todo esto en la línea 30, ejecutaremos el programa y veremos que la respuesta sigue igual **POR QUE???**.

La respuesta es muy sencilla, el Computador empieza ejecutando de izquierda a derecha comenzando por las multiplicaciones o divisiones y volviendo otra vez a la izquierda para ver si hay alguna suma o resta. Así pues la cifra 32 no se suma al 9 antes de ser dividido entre 5.

EJERCICIO 7-1

Escriba y ejecute un programa que convierta 65 grados Fahrenheit a grados Centígrados. La fórmula para esta conversión es la siguiente:

C = (F-32) * (5)/(9)

EJERCICIO 7-2

Quite los primeros paréntesis del Ejercicio 7-1 y ejecute de nuevo el programa.

EJERCICIO 7-3

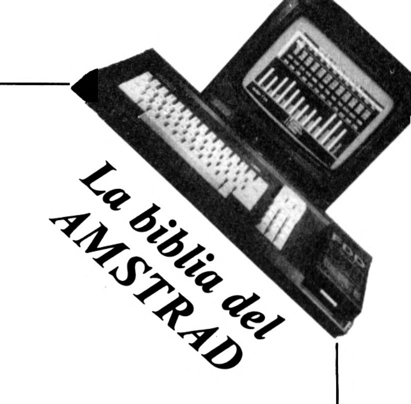
Ponga de nuevo los primeros paréntesis en la línea 30 y quite los segundos paréntesis de esa misma línea, ejecute el programa y verá los resultados.



EJERCICIO 7-4

En la ecuación que se muestra en este ejercicio, deberá escribir un programa, poniendo paréntesis entre las cifras que Vd. crea conveniente para que el resultado sea de 28. Escríbalo y ejecútelo.

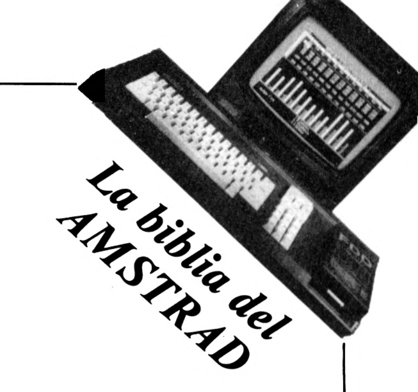
30 - 9 - 8 - 7 - 6 = 28



La biblia del AMSTRAD

Cristian Longhi

Grupo de trabajo Software, S. A.
Bailén, 20, 1.° Izq. 28005 MADRID
Depósito Legal: M.1.423-1986
Imprime: Gráf. FUTURA, Sdad. Coop. Ltda.
Distribuye: R.B.A. Promotora de Ediciones, S. A.
Travesera de Gracia, 56 - Atico. 1.ª
08006 - Barcelona
Teléfs.: (93) 200 82 56 - 200 80 45



CAPITULO 8

OPERADORES RACIONALES

Dado que se empieza a entrar en temas más importantes, como las sentencias **IF-THEN** y **GOTO** que permiten que su Computador tome decisiones para ejecutar estas acciones. Primero estudiaremos unos operadores más.

Los Operadores Racionales permiten al Computador el comparar un valor con otro, y constan de solamente tres tipos:

1.— El símbolo igual (=)
2.— El símbolo mayor que (>)
3.— El símbolo menor que (<)

Combinando estos tres símbolos se pueden obtener otros tres operadores más:

4.— El símbolo no igual a (< >)
5.— El símbolo menor que o igual a (<=)
6.— El símbolo mayor que o igual a (>=)

Añadiendo estos seis Operadores Racionales a los cuatro Operadores Matemáticos, los cuales los hemos cubierto en capítulos anteriores, más las nuevas sentencias **IF-**



THEN y **GOTO**, conseguiremos un gran sistema para comparar y calcular, lo que es el corazón principal de todo lo que viene a continuación.

La sentencia **IF-THEN** en combinación con los seis Operadores Racionales da una parte de acción de un sistema lógico.

Entrar y ejecutar el siguiente programa:

```
10 A = 5
20 IF A = 5 THEN 50
30 PRINT "A NO ES IGUAL QUE 5."
40 END
50 PRINT "A ES IGUAL QUE 5."
```

Una vez ejecutado este programa, en la pantalla saldrá el siguiente mensaje:

```
A ES IGUAL QUE 5.
```

Ahora examinaremos el programa línea a línea.

La línea 10 establece que el valor de la variable A es de 5.

La línea 20 es una sentencia **IF-THEN** que indica al Computador que salte a la línea 50 si el valor de la variable A es exactamente 5, saltándose las líneas que hay entre la 20 y la 50 y mostrando en pantalla el mensaje de la línea 50.

Si ahora modificamos la línea 10 para que se lea:

```
10 A = 6
```

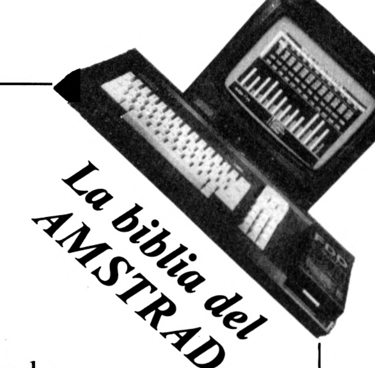
Ejecutaremos de nuevo el programa, obteniendo el siguiente mensaje en la pantalla:

```
A NO ES IGUAL QUE 5.
```

Examinando otra vez línea a línea veremos que:

En la línea 10 el valor de la variable A es ahora de 6.

La línea 20 comprueba el valor de la variable A, y dado que su valor no es de 5, no se saltará a la línea 50, sino que



el Computador se pasará a la siguiente línea la cual es la número 30.

La línea 30 le indica al Computador que imprima el mensaje de **A NO ES IGUAL QUE 5**., y no que el valor de la variable A no es 5, prosiguiendo el programa a la siguiente línea.

En la línea 40 se encuentra la sentencia **END**, la cual, de no encontrarse entre las líneas 30 y 50, el Computador seguiría a la línea 50 imprimiendo su mensaje en la pantalla, el cual obviamente crearía un conflicto con el contenido en la línea 30. Esto ha sido un ejemplo del uso de la sentencia **IF-THEN** relacionada directamente con el símbolo de **IGUAL.**

Las sentencias **THEN** y **GOTO** son opcionales en expresiones donde no se requiere el ir a una línea específica para la comprobación. Estas sentencias son muy útiles en sentencias muy largas de **PRINT** donde la comprobación dará un resultado que deberá ser mostrado en pantalla.

Por ejemplo:

```
20 IF X = 0 PRINT "X = 0"
```
(esto es correcto)

Pero si:

```
20 IF X = 0 100
99 END
100 PRINT "ES 100"
```
(esto es incorrecto)

En la línea 20 deberá haber lo siguiente:

```
20 IF X = 0 THEN 100
```

o

```
20 IF X = 0 GOTO 100
```

Ahora veremos si Vd. puede conseguir los mismos resultados usando el símbolo de «no es igual a»:



EJERCICIO 8-1

Escriba de nuevo el programa residente usando en la línea 20 el símbolo de «no igual a» (< >) en vez de el símbolo de «igual» (=), cambiando otras líneas que sean necesarias para conseguir el mismo resultado que en el programa del ejemplo.

EJERCICIO 8-2

Cambie el valor de la variable A de la línea 10 al valor de 6 dejando las restantes líneas intactas como aparecen en el Ejerciciio 8-1. Añada más líneas de programa para que el resultado nos indique que si A es mayor o menor que 5, y ejecútelo.

EJERCICIO 8-3

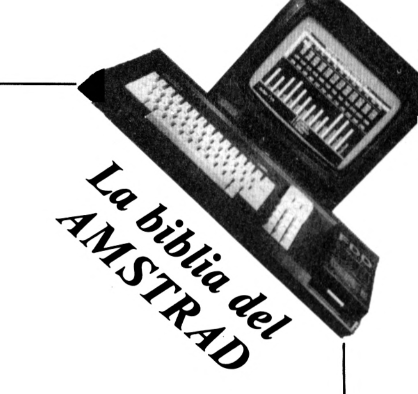
Cambie varias veces el valor de la variable A de la línea 10 del programa, ejecutándolo cada vez que lo cambie, hasta que vea que dicho programa le funciona correctamente.

La sentencia **IF-THEN** se la conoce como «Sentencia de Ramificación Condicional», ramificándose el programa a otra parte del mismo en la condición de comprobación de su contenido. Si la comprobación es falsa, simplemente el programa continúa a la siguiente línea.

La sentencia llamada **GOTO**, se la conoce también como «Sentencia de Ramificación Incondicional». Si en el programa que tenemos en máquina, sustituimos las líneas 40 y 80 con un **GOTO 99** y añadimos la línea 99:

99 END

El Computador una vez llegue a las líneas 40 y 80, incondicionalmente recibirá una orden de ir a la línea 99 finalizando la ejecución del programa. Practique varias veces con estos cambios para que así se habitúe al uso de la sentencia **GOTO** la cual será usada muy a menudo en el futuro.



CAPITULO 9

LA COMUNICACION CON EL COMPUTADOR

En este momento Vd. está cansado de estar cambiando el valor de la variable A de la línea 10, por lo que empezaremos a que dialogue con el Computador a través de la sentencia **INPUT**, la cual es muy sencilla, rápida y muy conveniente.

Añada las siguientes líneas al programa residente en máquina:

```
5 PRINT "QUE VALOR DESEA DAR:"
10 INPUT A
```

Ahora ejecute el programa, y el Computador le dará el siguiente mensaje en pantalla:

```
QUE VALOR DESEA DAR:
? █
```

El signo de interrogación que sale en la pantalla significa que es su turno para contestar y que el Computador le está esperando.

Para comprobar qué pasa, entre un número y verá que



el resultado es idéntico a cuando Vd. introducía ese mismo número en la línea 10 anteriormente. Ejecute varias veces este programa para que tome buen contacto con la sentencia **INPUT**.

Vamos a dar un toque de clase a la sentencia **INPUT**, modificando la línea 5:

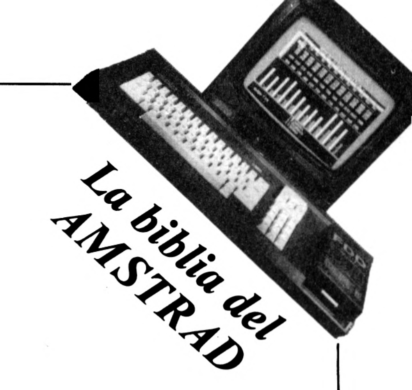
```
5 PRINT "QUE VALOR DESEA DAR:";
```

La diferencia que hay entre la primera línea 5 y la segunda, es que a esta última le hemos añadido al final un punto y coma. Si Vd. recuerda, anteriormente hemos usado el punto y coma con la sentencia **PRINT**, pero solamente en el medio de la sentencia para unir varias y así que apareciesen todas unidas y en la misma línea cuando saliesen en pantalla. En este caso se ha puesto un punto y coma al final para que el signo de interrogación aparezca en la misma línea y no en la línea siguiente como ocurría anteriormente. Una vez modificada la línea 5, ejecutaremos el programa y en la pantalla se podrá leer lo siguiente:

```
QUE VALOR DESEA DAR:? █
```

Hay que hacer notar, que Vd. no puede usar indiscriminadamente el punto y coma al final de una sentencia **PRINT**, y sólo se usa al final para unir dos líneas juntas en una impresión de una sola línea, y en este caso la sentencia **INPUT** es la que coloca el signo de interrogación. Más adelante veremos cómo dos líneas en las cuales se comience con una sentencia **PRINT**, pueden ser unidas colocando un punto y coma al final de la primera línea y salir en pantalla como una sola línea.

El Interpretador del Basic del Amstrad es capaz de poder dialogar en diferentes dialectos, siendo el primer dialecto que exploremos, a través de estos capítulos, los relacionados con la unión de las sentencias **PRINT** y **INPUT**. Para empezar, cambiaremos la línea 5 para que se lea de la siguiente forma:




```
5 INPUT "QUE VALOR DESEA DAR:"; A
```

Y a continuación eliminaremos la línea 10, tecleando solamente el número 10 y luego **ENTER**. Ejecute este programa y verá que el resultado es el mismo.

Resumiendo, esto es lo que hemos cambiado:

1.— La sentencia **PRINT** por la de **INPUT**.
2.— Ambas sentencias en una misma línea.
3.— Eliminar una línea extra.

Este tipo de sustituciones es muy práctico cuando se escriben programas largos, o se modifican, así como en su ejecución. Hasta el momento sólo hemos escrito programas cortos y de una respuesta, los cuales los hemos ejecutado tantas veces los hemos requerido y obtenido cada vez la palabra **Ready** al final de la ejecución. Para salirse de esta monotonía, escribiremos un programa basado en el que hicimos en el Capítulo 7, de conversión de grados Centígrados a Fahrenheit, pero modificándolo para que Vd. compruebe la utilidad de la sentencia **GOTO**.

Teclee lo siguiente:

```
NEW
CLS
10 REM * MODIFICACION DEL PROGRAMA DE CONVERSION DE GRADOS *
20 INPUT "QUE TEMPERATURA TENEMOS EN GRADOS CENTIGRADOS"; C
30 F = (9/5) * C + 32
40 PRINT C; "GRADOS CENTIGRADOS ="; F ;"GRADOS FAHRENHEIT."
50 GOTO 20
RUN
```

Con este programa Vd. verá que el Computador repite la misma pregunta una y otra vez hasta que Vd. se canse, dado que este tipo de máquinas son incansables. Modifique algún otro programa para que se ejecute de la misma forma, y así Vd. puede practicar con la sentencia **GOTO**.

Le recomendamos que si en este momento Vd. tiene alguna duda sobre los temas que hemos tratado en los diferentes Capítulos, vuelva sobre ellos, dado que los siguientes tratarán de conceptos y fundamentos más complicados.



CAPITULO 10

EL USO DEL COMPUTADOR COMO UNA CALCULADORA

Antes de entrar en la exploración de las partes más primordiales de nuestro Computador, veremos cómo lo podemos hacer trabajar en el modo de Calculadora. Si omitimos el número de la línea antes de ciertos comandos, el Computador los ejecutará imprimiendo los resultados en la pantalla y borrando el comando usado. Además este modo se puede usar aunque tengamos cargado un programa en memoria, sin molestarlo o incurriendo en errores.

Para poder pasar al modo de Calculadora, sólo necesitamos el signo del cursor en la pantalla y a partir de ese momento podremos calcular lo que queramos.

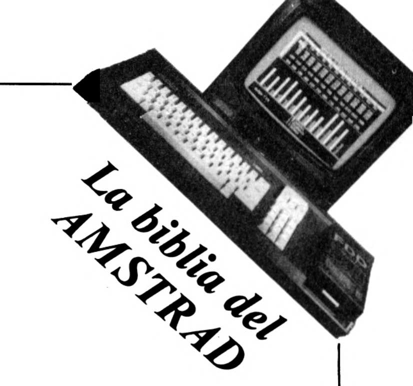
Ejemplo: ¿Cuánto es 3 por 4? Teclearemos lo siguiente:

```
PRINT 3 * 4
```

El resultado aparecerá en pantalla como:

```
12
```

Ejemplo: ¿Cuánto es 345 dividido entre 123?




```
PRINT 345 / 123
```

La respuesta será:

```
2.80488
```

Dedique unos minutos en poner problemas de rutinas matemáticas que a Vd. se le ocurran, usando el modo de Calculadora. Se podrá usar cualquier expresión aritmética que existe en un programa, en este tipo de modo de cálculo, incluyendo los paréntesis así como los cálculos en cadena, como A * B * C

Pruebe el siguiente problema:

```
PRINT (2/3) * (3/2)
```

La solución será:

```
1
```

Otro uso del modo de Calculadora, es el poder analizar algún problema de cálculo que tengamos en un programa, cuando la respuesta que nos ha dado el programa no es la adecuada. Así pues podemos hacer uso del Computador para saber dónde está el problema en la variable residente.

Ejemplo:

```
PRINT X  (El Computador mostrara el valor actual de la variable X.)
```

Otro caso es, cuando se ha almacenado en cada celdilla de la memoria, aunque Vd. no hubiese puesto nada dentro de ellas. Entrar y ejecutar esta instrucción en modo de Calculadora:

```
PRINT A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
```

La respuesta dependería de los valores que se han dado a esas variables, pero si Vd. desconecta el Computador y lo conecta otra vez, las variables estarán todas a 0.

Una de las características de este Computador, es el saber en cualquier momento cuánta memoria ha usado, dado



que todos los programas ocupan un espacio de esa memoria y está limitada al tamaño de la misma, por eso se creó este Comando de Memoria.

Para saber la cantidad de memoria usada, carge el programa que va a ejecutar y una vez lo tenga en memoria y aparezca el símbolo del cursor, teclee lo siguiente:

```
PRINT MEM
```

La respuesta que dará, será la memoria usada por dicho programa.

Para ver en la práctica de que tratamos, introduzcamos una línea de programa y luego veremos la cantidad de memoria que ha ocupado.

Entre la palabra **NEW** y luego la siguiente línea:

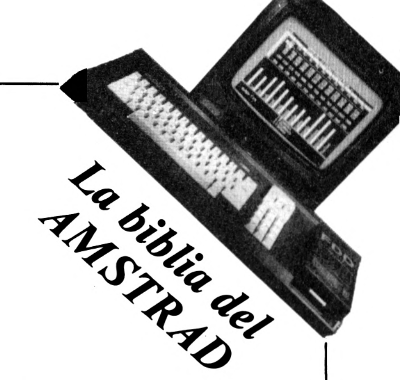
```
10 A = 25
PRINT MEM
```

El resultado será de 11, y la distribución de estos bytes será de la siguiente forma:

1.— Cada número de línea más su espacio correspondiente (sin importar el tamaño del número) ocupa 4 bytes, así como el retorno del carro al final de la línea ocupa 1 byte, haciendo un total de 5 bytes.

2.— Cada letra, número carácter o espacio usa un byte de memoria, así pues en el ejemplo (A = 25) la memoria ocupada será de 6 bytes. Si a los 5 bytes del número de línea más su espacio correspondiente más el retorno de carro le sumamos los 6 bytes de los caracteres, tendremos un total de 11 bytes de memoria usada por esa línea.

Practique Vd. un poco con este comando, escribiendo algunos de los programas anteriormente expuestos y viendo la cantidad de memoria que usan. No se olvide de te-



clear la palabra **NEW** antes de introducir un nuevo programa. Esta práctica le podrá ser de mucha ayuda en el futuro, dado que podrá retocar los programas para ahorrar parte de la memoria usada y dejar la mayor parte de ella libre para su ejecución.



CAPITULO 11

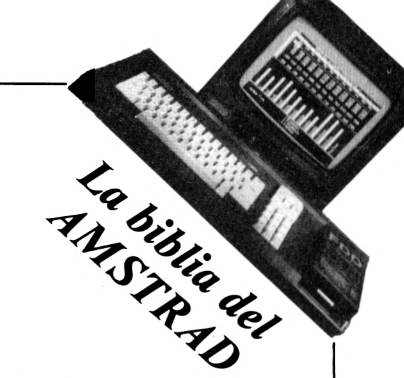
EL USO DEL ALMACENAMIENTO EN CASSETTE Y DISCO

Dentro de muy poco estaremos listos para escribir programas bastante largos y naturalmente no sería muy práctico el tenerlos que teclear cada vez que se quisieran ejecutar. Por esta razón, se ha incorporado a este Computador una grabadora reproductora de Cassettes (DATACORDER) para facilitar el almacenamiento de los programas y asimismo poderlos cargar en memoria cuando sea necesario.

Todos los controles de la operación de la Cassette se encuentran en unas direcciones residentes en ROM, las cuales son ejecutadas cuando se accede a ellas. Naturalmente no nos vamos a meter en la explicación de estas direcciones de memoria, ni lo que hacen, dado que la operación de manejo no lo requiere.

En el Basic del Amstrad hay una serie de comandos para el uso de la Cassette, los cuales son los siguientes:

SAVE. Este comando salva el contenido del programa o de datos que se encuentran en memoria a la Cassette.



LOAD. Carga de la Cassette, el contenido del programa o de los datos en memoria para que puedan ser ejecutados.

RUN. Carga y ejecuta el programa que contiene la Cassette.

SPEED WRITE 1. Salva el programa contenido en memoria a una velocidad de 2.000 baudios (velocidad rápida).

SPEED WRITE 0. Salva el programa contenido en memoria a una velocidad de 1.000 baudios (velocidad lenta).

CAT. Este comando sirve para sacar un catálogo del contenido de una cinta de Cassette.

Naturalmente Vd. podrá ampliar toda esta información, usando el manual de uso de su Computador en el cual viene todo muy bien definido, así como el uso de las teclas de su DATACORDER. En el caso de que Vd. usase una unidad de disco de tres pulgadas, el manual del usuario del disco le explica el uso del mismo.

Una de las recomendaciones que damos, es el usar siempre una cinta de Cassette virgen, o borrar con un borrador de cintas una usada antes de grabar el contenido de la memoria, dado que si no lo hace, hay casos que quedan datos de otros programas y pueden destruir su nuevo programa a la hora de cargarlos en memoria. Otra recomendación es que cuando tenga que salvar un programa bastante largo o importante, lo haga por duplicado en la misma cinta o en una segunda cinta. Dado que Vd. puede guardar programas uno detrás de otro en la misma Cassette, no necesitará tener gran cantidad de cintas, aunque es a gusto del usuario la forma de almacenamiento.

Practique con su DATACORDER salvando alguno de los programas que hemos estado escribiendo en Capítulos anteriores, y así se familiarizará con este tipo de operación que es de vital importancia.

Cualquier duda que Vd. tenga sobre esta operación, diríjase al manual de usuario de su equipo antes de destruir un programa que le ha costado horas el teclearlo.



CAPITULO 12

EL BUCLE FOR-NEXT

Una de las mayores diferencias entre un Computador y una Calculadora es la habilidad de hacer las mismas cosas cuantas veces uno quiera y a una velocidad altísima.

El bucle **FOR-NEXT** es de una abrumadora importancia para poder poner su Computador a trabajar en unas áreas de programación que serán exploradas a continuación.

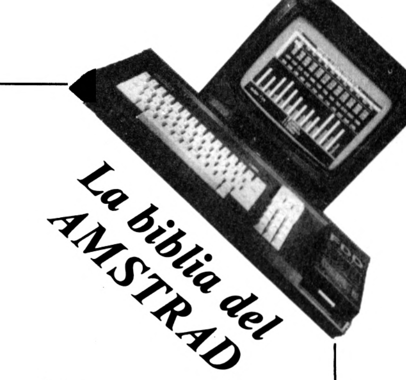
Para empezar con la explicación de este bucle, escribiremos una rutina muy familiar para Vd., pero antes limpiaremos la memoria tecleando la palabra **NEW**.

```
10 PRINT "AYUDA...MI COMPUTADOR SE HA VUELTO LOCO!!"
20 GOTO 10
RUN
```

Veremos que en la pantalla aparece continuamente la frase escrita en la línea 10

```
AYUDA...MI COMPUTADOR SE HA VUELTO LOCO!!
```

Esta frase no parará de salir,hasta que uno mismo quiera que pare, y para esto bastará con pulsar la tecla **ESC**.



Hemos creado lo que se llama un «Bucle sin fin» (recuerde cuando hemos usado la sentencia **INPUT** en programas anteriores). En la línea 20 tenemos una sentencia incondicional de **GOTO** que actúa sobre el Computador en forma cíclica entre las líneas 10 y 20.

Modifiquemos el programa anterior de la siguiente forma:

```
8 FOR N = 1 TO 5
10 PRINT "AYUDA...MI COMPUTADOR SE HA VUELTO LOCO!!"
20 NEXT N
30 PRINT "NO...ESTA BAJO CONTROL."
RUN
```

La línea:

```
AYUDA...MI COMPUTADOR SE HA VUELTO LOCO!!
```

Aparecerá en pantalla cinco veces, y luego saldrá la línea:

```
NO...ESTA BAJO CONTROL.
```

El bucle **FORT-NEXT** creado en las líneas 8 y 20, hace que el Computador entre en un ciclo a través de las líneas 8, 10 y 20 exactamente cinco veces para luego continuar con el resto del programa. Cada vez que el programa pasa por la línea 20 y encuentra la sentencia **NEXT N**, aumenta el valor de N exactamente 1, y el Computador incondicionalmente vuelve a la sentencia **FOR N** = para empezar el bucle de nuevo. Después del quinto pase por el bucle, éste se rompe y el programa continúa su ejecución.

LA FUNCION STEP

Hay veces que es necesario incrementar el bucle **FOR-NEXT** por un valor que no sea 1, teniendo que usar la función **STEP**.

Para demostrar su uso, cambiaremos la línea 8 para que se lea de la siguiente forma:

```
8 FOR N = 1 TO 5 STEP 2
RUN
```



En este caso, la línea 10 ha sido imprimida en pantalla solamente tres veces (cuando N = 1, N = 3 y N = 5), dado que el primer pase del programa, cuando se llegó a la sentencia **NEXT N**, el valor fue incrementado por 2 en vez de por 1. En el segundo pase por el bucle, N fue igual a 3, y en el tercer pase N fue igualado a 5.

Los bucles **FOR-NEXT** pueden ser incrementados por números decimales, positivos o negativos. La razón por la que se pueden usar números negativos se podrá demostrar a continuación con el cambio de la línea 8 del programa que deberá leerse de la siguiente forma:

```
8 FOR N = 5 TO 1 STEP -1
RUN
```

Los cinco pases por el bucle en pasos desde 5 a 1, es exactamente igual que si fuesen de 1 a 5, dado que la línea 10 sería impresa en pantalla cinco veces. Cambiando la sentencia **STEP** de –1 a –2.5 y ejecutando el programa otra vez, podrá observar que la línea 10 es imprimida dos veces. Cambie otra vez la línea 8 a **STEP –1**.

Modificación del bucle FOR-NEXT

Supongamos que queremos imprimir en pantalla cinco veces las líneas 10 y 30 intercalándose ambas líneas, piénselo y haga el cambio en el programa para conseguir este propósito.

Le daré una pista: pruebe a mover la línea de NEXT N en otra posición y verá los resultados.

CORRECTO, ha movido la línea 20 a la línea 40 y en su pantalla se puede leer lo siguiente:

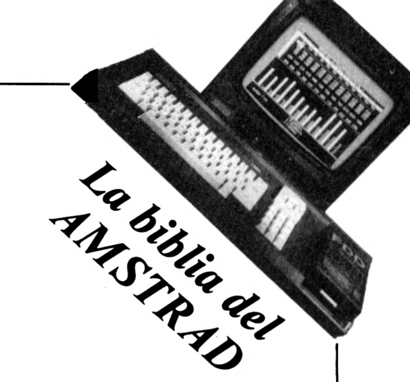
```
AYUDA...MI COMPUTADOR SE HA VUELTO LOCO!!

NO...ESTA BAJO CONTROL.

AYUDA...MI COMPUTADOR SE HA VUELTO LOCO!!

NO...ESTA BAJO CONTROL.

....etc etc tres veces mas.
```



Ejercite de nuevo con la siguiente modificación en el programa:

Qué cambios haría Vd. en el programa, para que la línea 10 se imprima cinco veces, y la línea 30 solamente tres. Modifíquelo y ejecútelo para ver los resultados:

El nuevo programa deberá leerse de la siguiente forma:

```
8 FOR N = 1 TO 5
10 PRINT "AYUDA...MI COMPUTADOR SE HA VUELTO LOCO!!"
20 NEXT N
25 FOR M = 1 TO 3
30 PRINT "NO...ESTA BAJO CONTROL."
40 NEXT M
```

Ahora tenemos un programa con dos bucles, el primero hace que la línea 10 aparezca cinco veces y el segundo bucle ordena que la línea 30 se imprima solamente tres veces. Hemos usado diferentes letras para los dos bucles, en el primero se ha tomado la letra N y en el segundo la letra M, pero Vd. podrá usar la que más le convenga. En este caso, podríamos haber usado la misma letra N para ambos bucles, dado que están totalmente separados, pero en la práctica cuando los programas son más largos conviene usar diferentes tipos de letras.

Ejecute otra vez el programa anterior para que entienda los principios fundamentales y las variaciones que hemos introducido.

No hay nada mágico en relación con el bucle **FOR-NEXT** dado que Vd., a lo mejor ha pensado alguna otra fórmula para conseguir los mismos resultados que si recuerda se trató en Capítulos anteriores. Así pues, pare un poco y piense en la manera de poder sustituir el **FOR** y el **NEXT** por otra cosa y que el resultado sea el mismo:



La solución es la siguiente:

```
8 N = 1
10 PRINT "AYUDA...MI COMPUTADOR SE HA VUELTO LOCO!!"
15 N = N + 1
20 IF N < 6 THEN 10
30 PRINT "NO...ESTA BAJO CONTROL."
```

El análisis de este programa es el siguiente: en la línea 8 hemos inicializado el valor de N, dándole un valor de 1, dado que antes de su inicialización, su valor pudiese ser cualquier número tomado de las otras líneas, y una vez ejecutamos el programa (run) ponemos todas las variables a 0.

En la línea 15, incrementamos el valor de N en 1 del que tenía anteriormente. La línea 20, usa uno de los operadores con el signo menor que para que compruebe que el nuevo valor de N está dentro de lo establecido, y si no el programa continúa su ejecución.

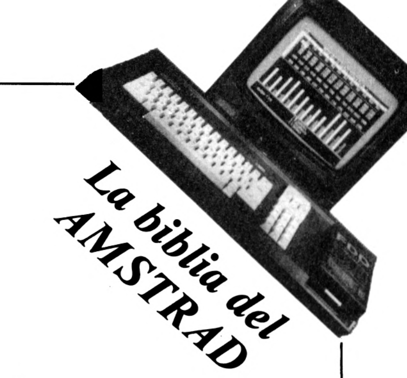
Notará que en este modo de incrementación y comprobación no se le ha mandado al programa que salte a la línea B, como en el caso del bucle **FORT-NEXT**. ¿Que pasaría si lo hiciésemos?

RESPUESTA: Estaríamos reinicializando el valor de N a 1 y crearíamos un bucle sin fin.

Lo opuesto a «incrementar» es «decrementar», así pues vamos a cambiar la línea 15 para que se lea de la siguiente forma:

```
15 N = N - 1
```

Efectúe otros cambios para que el programa funcione y ejecútelo para ver sus resultado.



RESPUESTA: Vd. debería haber cambiado las siguientes líneas:

```
8 N = 6
15 N = N - 1
20 IF N > 1 THEN 10
```

No es muy entretenido el estar haciendo lo mismo una y otra vez, y hay otras formas de poder demostrar las ventajas del bucle **FORT-NEXT**.

Supongamos que queremos sacar por pantalla, una tabla donde se muestre el tiempo que se tarda en ir desde Londres a San Diego, variando la velocidad de crucero del avión. Recordará la famosa fórmula **D = R * T,** pues esta tabla deberá mostrarnos la variación en tiempo, en una velocidad entre 200 Km/h. y 1000 Km/h en incrementos de 100 Km/h.

El programa será como sigue:

```
NEW
10 REM * TABLA DE TIEMPOS Y VELOCIDADES *
20 CLS
30 D = 6000
40 PRINT "        L O N D R E S   A   S A N   D I E G O   "
50 PRINT
60 PRINT "VELOCIDAD"; "HORAS";   "DISTANCIA (KLM)"
70 PRINT
80 FOR R = 200 TO 1000 STEP 100
90 T = D / R
100 PRINT R; T; D
110 NEXT R
```

Realmente es el mismo problema del Capítulo 5 pero repetido 9 veces para diferentes valores, y su pantalla dará un resultado como el siguiente:




```
L O N D R E S   A   S A N   D I E G O

VELOCIDAD        HORAS       DISTANCIA (KLM)

200              30          6000
300              20          6000
400              15          6000
500              12          6000
600              10          6000
700              8.57143     6000
800              7.5         6000
900              6.66667     6000
1000             6           6000
```

Analicemos el programa

En la línea 20 usamos el comando **CLS** para limpiar la pantalla.

La línea 30 inicializa el valor de D

La línea 40 imprime la cabecera a doble espacio solamente por estética.

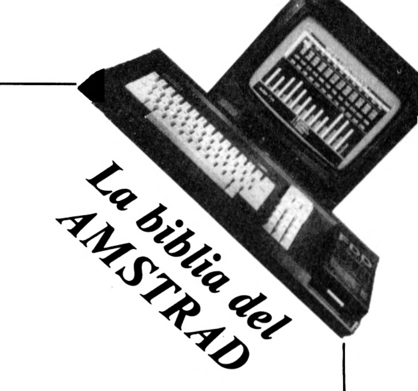
En las líneas 50 y 70 se usa la sentencia **PRINT** para espaciar los resultados.

La línea 60 imprime la información referente a los valores.

En la línea 80 se establece un bucle de **FOR-NEXT** completándolo con un **STEP** inicializando el valor de R a 200 y haciendo pases del bucle en incrementos de 100 hasta el final cuando se llegue a 1000, siendo la línea 110 la otra mitad del bucle.

En la línea 90 se encuentra la fórmula actual que calcula la respuesta. Y en la línea 100 se imprimen los tres valores de la respuesta al problema.

Tómese un descanso y piense si tiene alguna duda sobre este Capítulo con lo cual le recomiendo que salve este programa en cinta o disco dado que se usará en el próximo Capítulo.



CAPITULO 13

MAS SOBRE EL BUCLE FOR-NEXT

Si Vd. ha desconectado su Computador, deberá cargar de nuevo el programa último del Capítulo anterior, que lo debería haber salvado en cinta.

Una vez esté cargado el programa y comprobado que funciona, modificaremos el mismo para variar un poco los resultados.

Supongamos que en vez de un incremento de 100, deseamos un incremento de 50. Modifique este programa y ejecútelo para ver los resultados con este nuevo incremento.

La solución está en la línea 80 que deberá leerse de la siguiente forma:

```
80 FOR R = 200 TO 1000 STEP 50
```

Dado que una vez ejecutado la solución se muestra en pantalla a una velocidad muy rápida como para comprobarlo, pulsando la tecla **ESC** la pantalla se parará, volviéndose a reanudar pulsando cualquier otra tecla.



Para demostrar la importancia que tienen los «Bucles de tiempo», emplearemos otro truco muy interesante introduciendo un segundo programa de la siguiente forma:

Empiece entrando por el teclado:

```
9 END
```

Usaremos el espacio entre las líneas 1 y 8 para escribir y experimentar con el segundo programa.

El segundo programa será de la siguiente forma:

```
1 PRINT "NO SE MARCHE"
2 FOR X = 1 TO 4000
3 NEXT X
5 PRINT "EL PROGRAMA FINALIZARA"
RUN
```

La ejecución del programa tardará aproximadamente unos diez segundos, dado que el Computador ejecuta el bucle **FOR-NEXT** en 400 veces por segundo aproximadamente, significando que este retardo de tiempo lo puede Vd. controlar variando la línea 2.

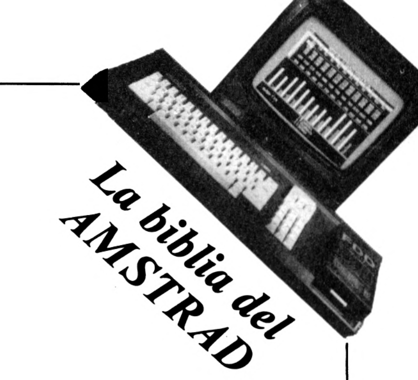
EJERCICIO 13-1

Usando el espacio entre las líneas 1 y 8, diseñar un programa donde le pregunte cuántos segundos de retardo desea Vd., permitiéndole entrar el número que desea, ejecutando el retardo y dando el resultado.

Cómo manejar listados largos

Como hemos visto anteriormente, para listar programas se usaba el comando **LIST**, el cual le imprimía en pantalla todo el listado del programa, pero Vd. llegará a tener programas bastantes grandes, y el listarlos le ocasionará bastantes problemas para seguirlos, por lo que a continuación le daremos la solución.

Entre las diferentes variaciones del comando **LIST** que a continuación le ponemos:



LIST 50 (listará solamente la línea 50)
LIST – 50 (listará todas las líneas hasta la 50)
LIST 50 – (listará desde la línea 50 hasta el final)
LIST 30-70 (listará solamente las líneas desde la 30 a la 70)
LIST. (listará la línea que está en ese momento)

Otro truco que Vd. deberá tener en cuenta, es que si por ejemplo tiene dos programas residentes en el Computador, puede ejecutar el primero con solamente entrar la palabra **RUN**, y para ejecutar el segundo sólo tiene que entrar el comando:

RUN ### (donde # # # es el número de línea donde comienza dicho programa)

Mientras tanto, tenemos un programa metido en la memoria del Computador con el cual veremos diferentes aspectos referentes a los «bucles de tiempo».

Primero borraremos la parte de prueba del programa escribiendo **DELETE 1-9 ENTER**, a continuación entre **LIST**. Lo que ha pasado es que se ha borrado desde la línea 1 a la 9 asemejándose este comando a lo que se puede hacer con el de **LIST**. La única variación entre uno y otro es que sólo se puede usar en **DELETE** lo siguiente:

```
DELETE ### (donde ### es el numero de linea)
DELETE -###
DELETE ###-###
```

Así que continuemos, y entremos la siguiente línea:

```
85 IF R = 600 THEN STOP
```

Esto hará que el programa se pare cuando la variable R llegue a la cifra de 600, apareciendo en la pantalla el mensaje de:

Break

Significando que en el programa se ha producido una ruptura de la ejecución, y pudiéndolo continuar con su eje-



cución, usando el comando **CONT**, sin alterarse para nada el programa mientras esté parado.

Por último crearemos un bucle de tiempo en el programa para así, poder estudiar el resultado en pantalla sin tener que meter el comando **STOP**. Teclee lo siguiente:

```
85 IF R <> 600 THEN 90
87 FOR X = 1 TO 4000
88 NEXT X
RUN
```

Maravilloso!!! funciona, mientras que R no sea igual a 600 el programa saltará sobre las 87 y 88, hasta que su valor llega a ser 600 entrando en acción el bucle de tiempo (líneas 87 y 88) haciendo que el programa se pare durante aproximadamente unos diez segundos.

EJERCICIO 13-2

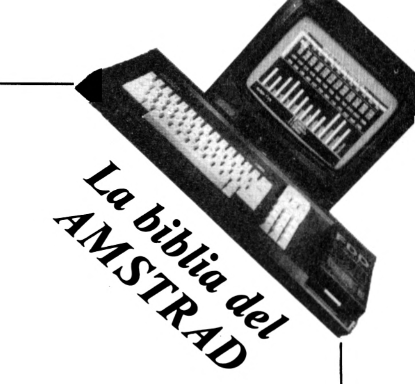
Modifique el programa residente en memoria para que la palabra Km/H aparezca debajo de VELOCIDAD, TIEMPO debajo de HORAS, y Km debajo de DISTANCIA.

EJERCICIO 13-3

Escriba y ejecute un programa que calcule e imprima (si tiene Vd. impresora), las retribuciones en base a: anualidad, mensual, semanal y diario, basado en 40 horas semanales, 12 meses al año, y en 52 semanas, con una retribución anual entre 500.000 pts y 2.500.000 pts. en incrementos de 100.000 pts. Documente su programa con sentencias REM.

EJERCICIO 13-4

Escriba y ejecute un programa donde le indique cuántos días debe Vd. trabajar, empezando con 100 pts al día y doblando este sueldo cada día que pasa, hasta que llega a ganar un millón. Debe incluir unas columnas donde apa-



rezcan el número del día, su ganancia diaria, y el total de ganancias a la fecha.

EJERCICIO 13-5

Si Vd. tiene un rollo de alambre de 1000 metros y desea cercar una parcela rectangular, determinar las dimensiones de largo y ancho donde se pueden cercar el máximo de metros cuadrados usando todo el alambre.

La fórmula para el área es: AREA = LARGO por ANCHO, o lo que es igual:
A= L * W

EJERCICIO 13-6

Una de las muchas ventajas de un Computador, es la asistencia en cálculos eléctricos por lo que probaremos si Vd. es capaz de crear un programa más sofisticado.

Con los valores que se dan en el esquema, escribir y ejecutar un programa donde se calculen los diferentes valores de una resistencia de carga que vayan de 1 a 20 Ohmios en incrementos de 1 Ohmio, imprimiendo las respuestas como siguen:

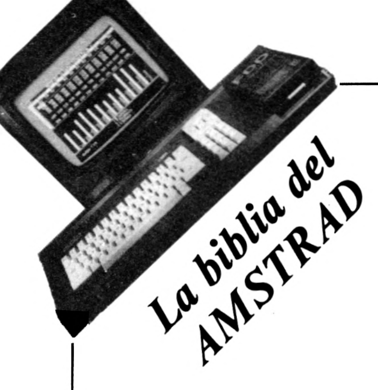
1.— Valor de la Resistencia de Carga (de 1 a 20 Ohmios).

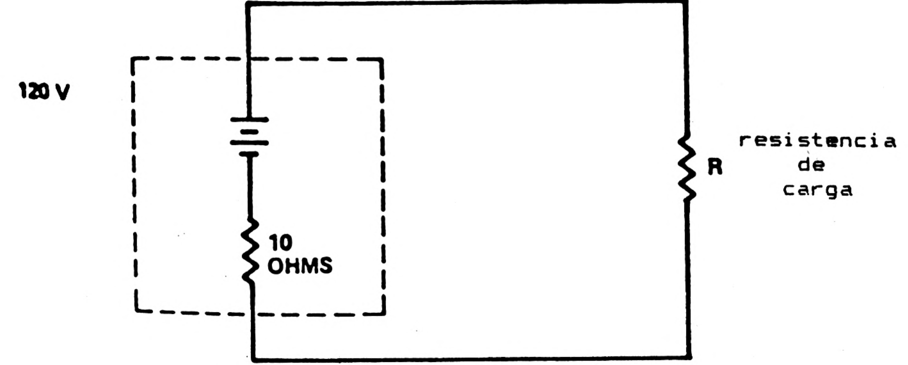
2.— Total de la potencia del circuito (la fórmula es: corriente del circuito al cuadrado por la resistencia del circuito)

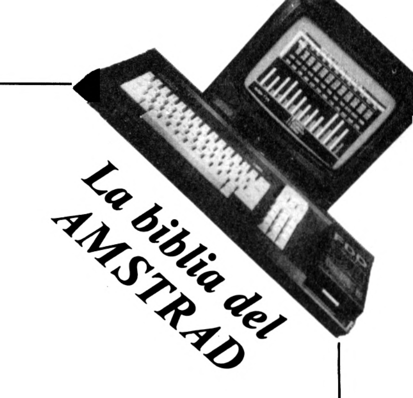
3.— Pérdida de potencia en origen (corriente del circuito al cuadrado por la resistencia en origen)

4.— Potencia de la resistencia de carga (corriente del circuito al cuadrado por la resistencia de carga)

Nota: la corriente del circuito se calcula dividiendo el voltage (120 V) por el total de la resistencia del circuito (10 Ohmios). Todo esto sigue la ley de OHMS que es: V = I * R y para la potencia P = I * V. Buena Suerte!!!!!!


La biblia del
AMSTRAD


120 V
10
OHMS
R
resistencia
de
carga



CAPITULO 14

FORMATEANDO CON EL TABULADOR

Después de un par de Capítulos un poco complicados, nos meteremos en uno un poco más sencillo. Dado que hasta ahora hemos aprendido tres formas de usar la sentencia **PRINT**, lo que a continuación explicaremos será lo siguiente:

1.— Se deberá poner entre comillas las sentencias con sus espacios correspondientes de lo que queremos que salga en pantalla.

2.— Separar los objetos de las sentencias **PRINT** con un punto y coma para que todo salga en una misma línea.

3.— Se usará la función TAB (la misma función de tabulación de una máquina de escribir), para tabular el texto y así tener una pantalla organizada.

Para demostrar esta última función, escribimos este pequeño programa:

```
10 PRINT TAB(5);"EL";TAB(20);"TOTAL";TAB(35);"GASTADO"
20 PRINT TAB(5);"PRESUPUESTO";TAB(20);"MES";TAB(35);"ES"
30 PRINT TAB(5);"CATEGORIA";TAB(20);"PRECIO";TAB(35);"UNIDAD"

RUN
```



EJERCICIO 14-1

Escriba un programa usando los tres formatos de **PRINT** con las formas que se le dan a continuación:

```
1.- PRINT "     ","       ",       "
2.- PRINT "                              "
3.- PRINT TAB( );"     ";TAB( );"       ";TAB( );"      "
```

Recuerde que cuando se usa el formato 1, las sentencias aparecerán en la pantalla una debajo de otra, así que tenga en cuenta esta recomendación.

Se pueden usar los TAB (nn) con un punto y coma o sin él, dado que no afecta para nada y el Computador empezará a contar los espacios de izquierda a derecha, pero es bueno recordar que cuando se imprimen en pantalla números o variables de números, el Computador dejará un espacio para los signos + y –. Escriba y ejecute el siguiente programa:

```
10 A = 3
20 B = 5
30 C = A + B
40 PRINT TAB(10); "A";TAB(20); "B"TAB(30); "C"
50 PRINT TAB(10); A;TAB(20); B;TAB(30); C
```

El resultado aparecerá en la pantalla de la siguiente forma:

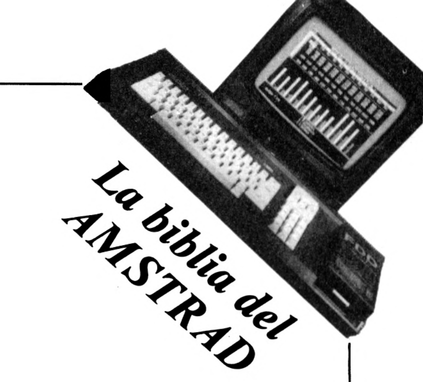
```
A                  B                  C
 3                  5                  8
```

Ahora cambiaremos la línea 20 para que se lea:

```
20 B = -5

RUN
```

¿Puede comprobar la diferencia que hay?



DIVIDIR LINEAS LARGAS

Un problema que se le puede presentar, es cuando necesita imprimir un gran número de cabeceras o respuestas en una misma línea, pero no hay sitio en la línea del programa. Para demostrarlo, escribiremos el siguiente programa:

```
10 A = 1
20 B = 2
30 C = 3
40 D = 4
50 E = 5
60 F = 6
70 G = 7
80 H = 8
90 I = 9
100 J = 10
200 PRINT "A"; TAB(5); "B";TAB(10); "C";TAB(15); "D";
210 PRINT TAB(20); "E";TAB(25); "F";TAB(30); "G";
220 PRINT TAB(35); "H";TAB(40); "I";TAB(45); "J"
300 PRINT A; TAB(5); B;TAB(10); C;TAB(15); D;TAB(20);
310 PRINT E;TAB(25); F;TAB(30); G;TAB(35); H;TAB(40);
320 PRINT I;TAB(45); J
RUN
```

El punto y coma al final de cada línea del programa hace que en la salida por pantalla aparezca todo en una misma línea, pero si le quitamos el punto y coma nos hará un retorno de carro al final de la misma.

EJERCICIO 14-2

Escriba de nuevo el programa del Ejercicio 13-3 para incluir el «PRECIO HORA» en la salida. Usar la función TAB para crear una tabla de 5 columnas.

EJERCICIO 14-3

Modifique la respuesta del problema 13-6, usando la función TAB para que en la pantalla aparezca incluida una quinta columna con la resistencia interna.



USO DE LA IMPRESORA DE PAPEL

Hasta ahora se ha tratado los diferentes modos de la sentencia **PRINT** para que el resultado saliese por la pantalla. A continuación le mostraremos cómo los resultados pueden salir por la impresora (si tiene Vd. una instalada a su Computador). En el caso de que no disponga de una, sáltese esta sección dado que no le será de ninguna utilidad.

Conecte la impresoa, póngala en línea con el Computador, y teclee la siguiente linea:

```
10 PRINT#8 "LA IMPRESORA FUNCIONA!!!"
```

Vea que hemos cambiado la palabra **PRINT** por la de **PRINT**#8 dado que el canal 8 es el de la impresora.

En el caso de que no salga nada por la impresora, compruebe la instalación de la misma y pruebe de nuevo el imprimir la línea editada. Una vez le funcione todo, escribiremos otro programa para que así Vd. vea los resultados.

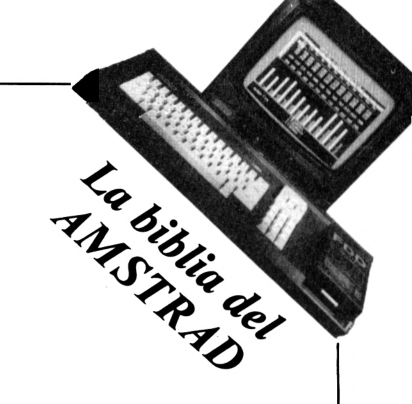
```
NEW
10 FOR X = 1 TO 100
20 PRINT#8 X;
30 NEXT X
RUN
```

Vea como la impresora le formatea la hoja en pequeñas columnas, dado que hemos puesto un punto y coma al final de la línea 20.

El uso de la función **TAB** en unión con la impresora, es muy recomendable, dado que Vd. puede formatear las hojas de salida a sus necesidades, pudiendo usar números en TAB (n=0–255), dependiendo del tipo de letra a usar y el ancho de su impresora.

Para ver cómo la función **TAB** actúa sobre un programa con salida por impresora, escribiremos y ejecutaremos el siguiente programa.

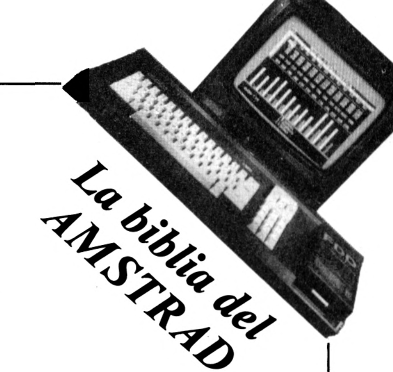
(Continuará)



La biblia del AMSTRAD

Cristian Longhi

Grupo de trabajo Software, S. A.
Bailén, 20, 1.º Izq. 28005 MADRID
Depósito Legal: M.1.423-1986
Imprime: Gráf. FUTURA, Sdad. Coop. Ltda.
Distribuye: R.B.A. Promotora de Ediciones, S. A.
Travesera de Gracia, 56 - Atico. 1.ª
08006 - Barcelona
Teléfs.: (93) 200 82 56 - 200 80 45




```
NEW
10 PRINT#8 TAB(20) "LISTIN TELEFONICO"
20 PRINT#8
30 PRINT#8 TAB(15) "NOMBRE"; TAB(45) "NUMERO TELEFONICO"
40 PRINT#8
50 INPUT "NOMBRE DE LA PERSONA"; A$
60 INPUT "NUMERO TELEFONICO"; B$
70 PRINT "MUCHAS GRACIAS"
80 PRINT#8 TAB(15) A$; TAB(45) B$
90 INPUT "ALGUN OTRO NOMBRE (S/N)"; Q$
100 IF Q$ = "S" THEN 50
RUN
```

En el caso de que el papel de su impresora fuese más estrecho, deberá usar otra numeración en las funciones **TAB** para formatear el texto en la hoja.

Otra cosa que puede Vd. hacer con su impresora, sería el listar su programa, para lo cual sólo debe teclear la palabra **LIST#8 Y ENTER**, y verá salir su listado.



CAPITULO 15

BUCLE AVANZADO DE FOR-NEXT

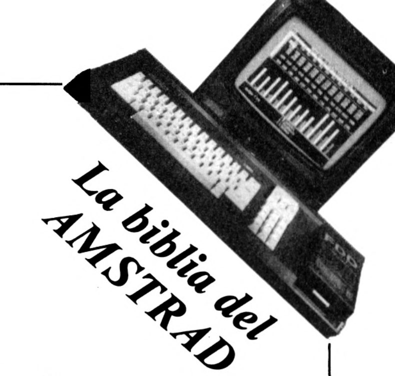
Como verá no hemos terminado con el dichoso bucle **FOR-NEXT**, dado que ahora lo trataremos diferentemente.

Entre el siguiente programa:

```
NEW
10 FOR A = 1 TO 3
20 PRINT "BUCLE A"
30 FOR B = 1 TO 2
40 PRINT " ", "BUCLE B"
50 NEXT B
60 NEXT A
RUN
```

El resultado sería esto:

```
BUCLE A
                    BUCLE B
                    BUCLE B
BUCLE A
                    BUCLE B
                    BUCLE B
BUCLE A
                    BUCLE B
                    BUCLE B
```



Este programa ha demostrado la utilidad del bucle **FOR-NEXT** para poder formatear una pantalla y que los resultados salgan ordenados.

Analicemos el programa paso a paso:

La línea 10 establece un bucle **FOR-NEXT** llamado A y lo direcciona para que se ejecute tres veces.

En la línea 20 está la sentencia **PRINT** con el texto **«BUCLE A»** pudiéndolo identificar el programa para sacarlo en pantalla.

La línea 30 establece el segundo bucle llamado B y lo direcciona para que se ejecute dos veces.

La acción de la línea 40 es la de imprimir los dos conceptos, de los cuales el primero está en blanco seguido por una coma para su posicionamiento del segundo con el texto de **«BUCLE B»**.

La línea 50 completa el bucle «B» y devuelve el control a la línea 30 para la cantidad de ejecuciones del bucle «B» como lo mandó dicha línea.

En la línea 60 finaliza el primer paso del bucle «A» mandando el control a la línea 10.

Muy bien, ahora veremos qué pasa cuando modificamos una línea. Empezaremos cambiando la línea 10 para que se lea de la siguiente forma:

```
10 FOR A = 1 TO 5
RUN
```

El resultado ha sido que el **BUCLE A** se ha impreso cinco veces y el **BUCLE B** diez, dos veces por cada paso del bucle «A». Ahora cambiemos la línea 30.

```
30 FOR B = 1 TO 4
RUN
```

Lo que tenemos ahora es que el **BUCLE A**, ha salido cinco veces y el **BUCLE B** veinte veces. Si tiene dificultad en contar los pasos, nada más que tiene que pulsar una sola vez la tecla de ESC y se parará la pantalla, reanudándose con solo pulsar cualquier otra tecla.



El siguiente paso es el hacer unos cambios para que vea lo que pasa cuando se varía el programa.

Cambiaremos las líneas siguientes:

```
50 NEXT A
60 NEXT B
RUN
```

En la pantalla del Computador le saldrá un mensaje de ERROR que dice lo siguiente:

```
Unexpected NEXT in 60
```

Examinando el programa veremos rápidamente que el bucle B no puede actuar con el bucle A, dado que tenemos parte de la sentencia **FOR** del bucle B dentro del bucle A, pero la sentencia **NEXT** se encuentra fuera. Así pues modifique su programa para que funcione poniéndolo en su estado original.

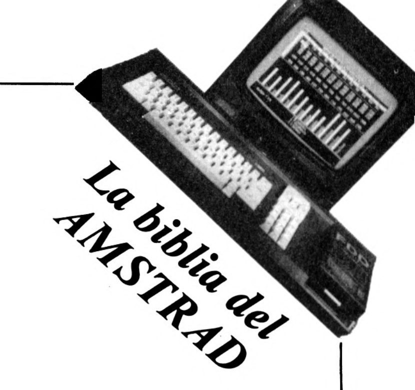
Seguiremos con nuestro programa añadiendo otras líneas para demostrar que la condición de división de los bucles es perfectamente legal. Así pues, añadiremos las siguientes líneas:

```
50 NEXT B
55 IF A = 2 GOTO 100
60 NEXT A
99 END
100 PRINT "A ES IGUAL A DOS...FIN"
RUN
```

Si Vd. ha modificado el programa anteriormente, no tendrá que meter las líneas 50 y 60.

EJERCICIO 15-1

Entre el programa original que se encuentra al principio de este Capítulo, el cual contiene un bucle B anidado con el bucle A. Haga las modificaciones necesarias a este programa para que un nuevo bucle C esté anidado con el bucle B y que se imprima **BUCLE C** cuatro veces por cada paso del bucle B.



EJERCICIO 15-2

Modifique el programa residente, que corresponde a la solución del ejercicio 15-1, añadiendo las líneas necesarias para que un nuevo bucle D sea anidado, o intercalado, con el bucle C, y que aparezca la palabra **BUCLE D** cinco veces por cada paso del bucle C.



CAPITULO 16

LA FUNCION DE NUMEROS ENTEROS

No tiene por que asustarse con el nombre de esta función, dado que un Número Entero puede ser cualquier número, como –5, 0 ó 3. La función de «Números Enteros» se representa como INT (X), permitiéndonos el redondear cualquier número, sea grande o pequeño, positivo o negativo.

Entre el siguiente programa:

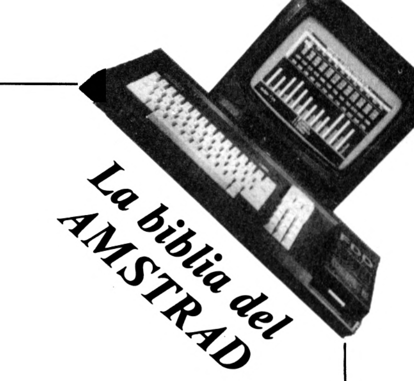
```
NEW
30 X = 3.14159
40 Y = INT (X)
70 PRINT "Y = "; Y
RUN
```

El resultado que saldrá en pantalla será;

```
Y = 3
```

Significando que la cifra 3.14159 la ha redondeado al número 3.

Modifiquemos la línea 30:




```
30 X = -3.14159
RUN
```

El resultado que tenemos ahora es:

```
Y = -4
```

Naturalmente el programa ha redondeado la cifra al valor inferior el cual es –4, dado que esta función siempre rendondea a negativo o hacia abajo.

Examinando línea a línea tendremos lo siguiente:

En la línea 30 coloca un valor a X (o otra letra si Vd. lo desea) igual al valor que hemos seleccionado, en este caso el valor de «PI».

La línea 40 encuentra el valor del **INTEGER** (Número Entero) de la línea anterior, y le asigna un nombre variable, en este caso Y.

La línea 70 imprime en pantalla la identificación (Y =) seguido por su valor.

A continuación jugaremos un poco con la combinación del bucle **FOR-NEXT** y con la función **INTEGER** (Número Entero).

Cambie el programa para que se lea de la siguiente forma:

```
30 X = 3.14159
40 Y = INT (X)
50 Z = X - Y
60 PRINT "X = "; X
70 PRINT "Y = "; Y
80 PRINT "Z = "; Z
RUN
```

El resultado que hemos obtenido es:

```
X = 3.14159
Y = 3
Z = .14159
```

En este caso se ha dividido el valor de X entre su Entero (**INTEGER**) y le hemos llamado Y y a su decimal Z.

Hay una forma de controlar la precisión de los resultados, e implica el redondear la fracción de los espacios decimales de un número, y forzar al Computador para que imprima solamente los dígitos que han sido redondeados.

Entre el siguiente programa para demostrarlo:

```
NEW
10 X = 3.14159
20 X = X + .0005
30 X = INT (X * 1000) / 1000
40 PRINT X
```

El resultado será de:

```
3.142
```

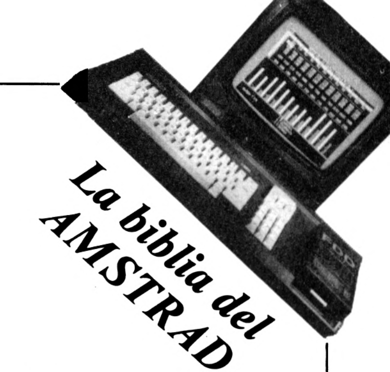
Pruebe con otros valores de X pero asegurándose de que no sean demasiado grandes para que los pueda manejar la función **INT**.

Ahora probaremos el usar el bucle **FOR-NEXT** para poder dividir el número en más lugares, para eso entraremos el siguiente programa:

```
NEW
30 X = 3.14159
40 Y = INT (X)
50 Z = X - Y
60 PRINT "X = ";X
70 PRINT "Y = ";Y
80 PRINT "Z = ";Z
90 M = Z * 10
100 L  = INT (M)
110 PRINT "L = ";L
RUN
```

El resultado será:

```
X = 3.14159
Y = 3
Z = 0.14159
L = 1
```



Lo siguiente que debemos hacer es añadir las siguientes líneas:

```
95 FOR A = 1 TO 5
120 M = M - L
130 M = M * 10
140 NEXT A
RUN
```

El resultado de esta nueva versión será el siguiente:

```
X = 3.14159
Y = 3
Z = 0.14159
L = 1
L = 4
L = 1
L = 5
L = 9
```

Para poder seguir la ejecución del programa, se pueden insertar líneas de **PRINT** y así nos saldrá en pantalla paso a paso lo que va sucediendo. Por ejemplo podríamos insertar las líneas siguientes:

```
5 MODE 2
92 PRINT ,"#92 M = "; M
97 PRINT ,"#97 A = "; A
125 PRINT , , "#125 M = "; M
135 PRINT , , "#135 M = "; M
RUN
```

Como Vd. podrá notar, en la línea 5 hemos usado un comando nuevo, el cual lo único que hace es cambiar el formato de pantalla a 80 carácteres por línea, esto se explica perfectamente en su manual de usuario.

EJERCICIO 16-1

Entre este sencillo programa para poder averiguar el área de un círculo.




```
NEW
10 P = 3.14159
20 PRINT "RADIO", "AREA"
30 PRINT
40 FOR R = 1 TO 10
50 A = P * R * R
60 PRINT R, A
70 NEXT R
RUN
```

Dado que la solución a este programa la da con muchos decimales, y hay personal que nada más están interesadas en números simples, modifique el programa para suprimir todos los números a la derecha del punto decimal.

EJERCICIO 16-2

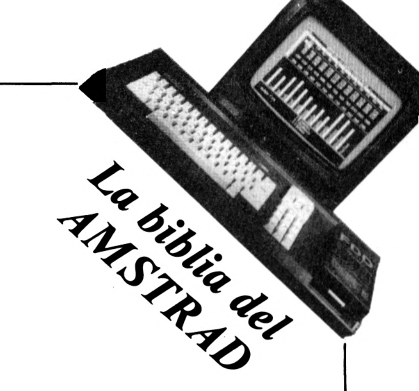
Como en este momento Vd. es un experto en estos cálculos, cambie la línea 55 para que cada valor de AREA esté redondeado con una precisión de un decimal. Por ejemplo:

```
    RADIO                    AREA
     1                       3.1

etc...
```

EJERCICIO 16-3

Sigamos complicando más las cosas. Cambie otra vez la línea 55 para que cada valor del AREA esté redondeado con una precisión de dos decimales.



CAPITULO 17

MAS SENTENCIAS DE RAMIFICACION

Para comenzar con este Capítulo, entraremos el siguiente programa:

```
NEW
10 INPUT "ENTRE UN NUMERO DEL 1 AL 5"; N
20 IF N = 1 GOTO 110
30 IF N = 2 GOTO 130
40 IF N = 3 GOTO 150
50 IF N = 4 GOTO 170
60 IF N = 5 GOTO 190
70 PRINT "EL NUMERO QUE Vd. HA ENTRADOS NO ESTA ENTRE EL 1 Y 5"
99 END
110 PRINT "N = 1"
120 END
130 PRINT "N = 2"
140 END
150 PRINT "N = 3"
160 END
170 PRINT "N = 4"
180 END
190 PRINT "N = 5"
```

Ejecute este programa varias veces para ver si lo entiende y así poder continuar.

De cualquier forma este programa es muy interesante para examinar los valores de una variable **N** y ordenar al



COMPUTADOR que pare cuando se ha ejecutado. Hay infinidad de maneras para ramificar, pero queremos usar la sentencia **ON-GOTO** que ahorra muchas líneas de programa, así como memoria en su Computador. Examinemos la sentencia **ON-GOTO** después de modificar este programa.

Borre con el comando **DELETE** las líneas 20, 30, 40, 50 y 60, y entre una nueva línea:

```
20 ON N GOTO 110,130,150,170,190
RUN
```

Verá que el resultado de la ejecución de este programa es el mismo que el anterior, con la diferencia de que nos hemos ahorrado cuatro líneas.

Dado que hay muchos trucos para sacar más jugo a la sentencia **ON-GOTO**, pondremos el siguiente ejemplo: Si queremos hacer una ramificación de quince diferentes localizaciones, pero no queremos escribir esa cantidad de números en la línea de **ON-GOTO**, podemos ahorrar muchas líneas haciendo lo siguiente:

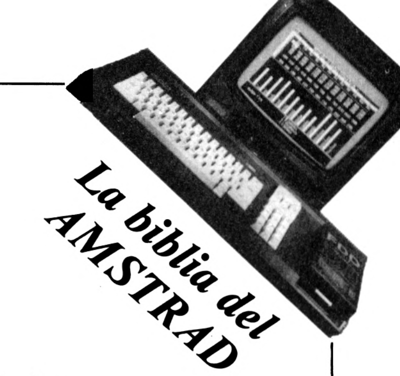
```
20 ON N GOTO 110, 130, 150, 170, 190
25 ON N - 5 GOTO 210, 230, 250, 270, 290
30 ON N - 10 GOTO 310, 330, 350, 370, 390
```

Rellenando el programa con las respuestas apropiadas.

En la línea 25 será necesario el restar 5 del número que hemos dado a la variable N, dado que cada nuevo **ON-GOTO** empieza contando desde el número 1. En la línea 30, dado que hemos provisto de entradas entre 1 y 10, restaremos 10 de la entrada a la variable N para convertirlo del 11 al 15. Usando la setencia **ON-GOTO** hemos programado en tres líneas donde de la otra forma usaríamos quince.

Una función llamada SGN (X)

La función **SGN** examina cualquier número para ver si es negativo, cero, o positivo, siendo una función muy sim-



ple. Para podernos meter en ella, vamos a crear una función **SGN** en una **SUBRUTINA**.

¿Qué es una **SUBRUTINA**?

Una Subrutina es un programa (o rutina) muy corto, pero importante, que está dentro de otro programa más grande para una necesidad muy especial. En Basic se almacenan muchas Subrutinas para posteriormente poderlas llamar con un juego simple de letras.

Para demostrar lo que es y cómo funciona una Subrutina, escribiremos el siguiente programa:

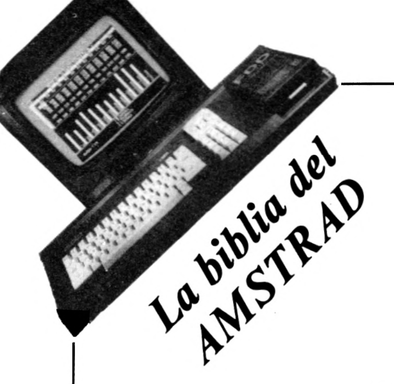
```
NEW
30000 END
30800 REM * SGN(X) * ENTRADA X, SALIDA T = -1,0, OR +1 *
30810 IF X < 0 THEN T = -1
30820 IF X = 0 THEN T = 0
30830 IF X > 0 THEN T = +1
30840 RETURN
```

Para llamar a una Subrutina se deberá usar la sentencia **GOSUB**, que direcciona al Computador para que se desplace a la línea indicada, la ejecute y luego volver (**RETURN**) a la línea de la sentencia **GOSUB**. En unas sencillas palabras, es una sentencia de ida y vuelta, lo contrario de la sentencia GOTO, que es solamente de ida. Para demostrarlo, usaremos la línea 20:

```
20 GOSUB 30800
```

Ahora combinaremos las sentencia **GOSUB** y **SGN** usándolas como subrutinas.

```
10 INPUT "ENTRE UN NUMERO CUALQUIERA"; X
20 GOSUB 30800
30 ON T +2 GOTO 50, 60, 70
45 END
50 PRINT "EL NUMERO ES NEGATIVO."
55 END
60 PRINT "EL NUMERO ES CERO."
65 END
70 PRINT "EL NUMERO ES POSITIVO,"
RUN
```



Examinando el programa, veremos lo siguiente:

La línea 10 le pide un número

En la línea 20 le manda al Computador que se dirija a la línea 30800 con la sentencia **GOSUB**

Una vez en la línea 30800 se ejecuta la subrrutina de la línea 30800 a la 30840.

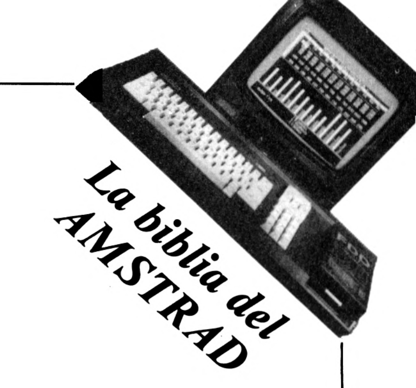
En la línea 30840 se encuentra un **RETURN** que devuelve el control a la línea 20 para que prosiga el programa

La línea 30 contiene una sentencia **ON-GOTO** pero suma 2 al valor de su variable, en este caso «T», indicando que si T es –1 diríjase a la línea 50, si es cero, a la línea 60, y si es +1, a la línea 70.

Las líneas 45, 55, y 65 son rutinas de bloque de protección.

EJERCICIO 17-1

Quite del programa todas las líneas de la subrrutina, y usando la función **SGN** consiga los mismos resultados que cuando se usaba la subrutina. (T = SGN(X)).



CAPITULO 18

NUMEROS ALEATORIOS

Un número Aleatorio es un número con un valor impredicible. El Generador de Números Aleatorios crea una serie de números diferentes, expresándolo de la siguiente manera:

```
N = RND (X)
```

En esta función, N es el número aleatorio, **RND** es la abreviación del símbolo de un aleatorio, y X es el número de control que puede estar o entre paréntesis, o se le puede llamar como una variable desde el programa.

Entre el siguiente programa:

```
NEW
40 FOR N = 1 TO 10
50 PRINT RND (0)
60 NEXT N
RUN
```

En la ejecución de este programa, ha podido observar que cada vez aparece un número diferente, que todos los números estaban entre el 0 y el 1, y que los números pequeños eran expresados en anotación exponencial.



Si modificamos el programa cambiando las líneas 40 y 50, tendremos lo siguiente:

```
40 FOR N = 1 TO 90
50 PRINT RND(0);
60 NEXT N
RUN
```

El punto y coma que se ha colocado al final de la línea 50, ha sido para que aparezcan más números en la pantalla de una sola vez, y se ha aumentado el bucle **FOR-NEXT** en 90 pases.

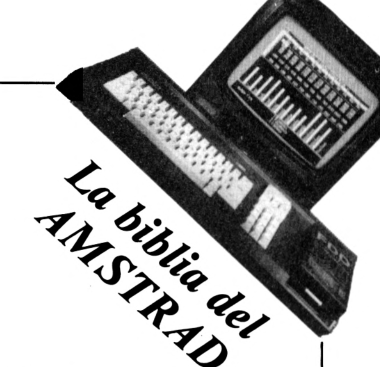
Para que Vd. vea lo que hacen los números aleatorios, escribiremos un programa bastante largo el cual una vez lo ejecute se examinará su contenido.

```
NEW
10 INPUT "CUANTAS VECES DESEA TIRAR LA MONEDA"; F
20 CLS
30 PRINT "ESPERE UN POCO MIENTRAS YO LA TIRO..."
40 FOR N = 1 TO F
50 X = RND(2)
60 ON X GOTO 90, 110
70 PRINT "SE HA EQUIVOCADO, NO HA SIDO NI 1 NI 2."
80 END
90 H = H + 1
100 GOTO 120
110 T = T + 1
120 NEXT N
130 PRINT "CARAS", "CRUCES", "TIRADAS TOTALES"
140 PRINT H, T, F
150 PRINT 100 * H / F; "%", 100 * T / F; "%"
```

Compruebe que el programa está escrito correctamente, y una vez comprobado podrá ejercutarlo con el comando **RUN**.

Dado que ha visto cómo funciona, empezaremos a examinar dicho programa:

En la línea 10, le pide el número de tiradas.
La línea 20 limpia la pantalla.
La línea 30 le pone una sentencia de espera.



En la línea 40 comienza un bucle de **FOR-NEXT** que se ejecuta F veces.

La línea 50 es el generador de **RND** (X), al cual le hemos mandado que genere números enteros entre el 1 y el 2.

En la línea 60 hay una comprobación **ON-GOTO**, mandando X = 1 a la línea 90, y X = 2 a la línea 110.

Las líneas 70 y 80 se usan como líneas por defecto con lo cual si X no es igual a 1 ó 2 aparecerá un mensaje y finalizará el programa. Esto no ocurrirá pero insista para probarlo.

La línea 90 es el contador de H, dado que cada vez que la sentencia **ON-GOTO** hace una comprobación manda el control a esta línea incrementando en uno y siguiendo contando.

La línea 100 manda el control a la línea 120 cuando la primera sentencia **NEXT N** es ejecutada hasta que el número de pase es igual a F, devolviendo el control a la línea 130. Hasta que esto ocurra, la sentencia **NEXT N** mandará el control a la línea 40.

La línea 50 genera otro número aleatorio.

La línea 60 manda el control a la línea 110.

La línea 110 lleva el control del número de veces que ha salido «Cruces».

La línea 120 pasa el control a la línea 130 cuando la última «N» es usada.

La línea 130 imprime las cabeceras.

La línea 140 imprime los valores de H, T, y F

La línea 150 calcula e imprime los porcentajes de «caras» y de «cruces».

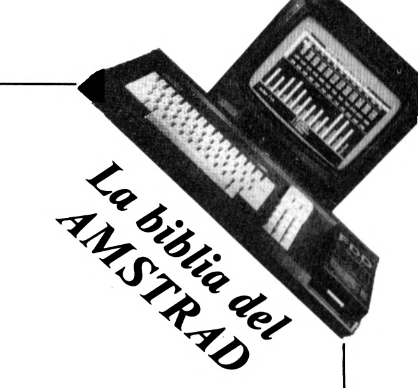
TAmbién hay la posibilidad de generar más de un número aleatorio, usando varios generadores dentro del mismo programa. Para demostrar este punto, crearemos un programa del juego de los «DADOS», donde tenemos dos dados con seis caras cada uno, teniendo las caras unos puntos que van del 1 al 6 respectivamente. Cuando se tiran los dos dados los puntos que aparecen en sus caras supe-



riores se suman, siendo el 2 la suma más inferior, y el 12 la superior. Se creará dos Generadores de Números Aleatorios para cada dado con un rango del 1 al 6 y llamando a un dado «A» y al otro «B».

```
NEW
50 A = RND(6)
60 B = RND(6)
70 N = A + B
80 PRINT N
RUN
```

Practique con este juego varias veces para ver sus resultados, y naturalmente se podrá divertir mucho con sus amigos.



CAPITULO 19

LAS SENTENCIAS REAC, DATA, Y RESTORE

Hasta ahora hemos aprendido a entrar números en un programa por dos métodos diferentes, siendo el primero la construcción de un valor en un programa

```
10 A = 5
```

y el segundo, usando la sentencia **INPUT** para entrar un número deseado.

```
10 INPUT A
```

El tercer método, el cual será el que trataremos en este Capítulo, es el que usa la sentencia DATA.

Para empezar la explicación, entrar el siguiente programa:

```
NEW
10 DATA 1, 2, 3, 4, 5
20 READ A, B, C, D, E
30 PRINT A, B, C, D, E
RUN
```

La sentencia **DATA**, de alguna manera, al primer méto-



do en que la línea de **DATA** es parte del programa, la diferencia que se encuentra es que cada línea de **DATA** contiene una cantidad de números o datos, todos ellos separados por una coma.

La sentencia **READ** es necesaria para poder leer cada parte de la sentencia **DATA**, pudiendo leer cada sección que esté separada por una coma, empezando a leer siempre de izquierda a derecha y empezando por la primera línea de **DATA**, sin importar donde se encuentre en el programa, así pues Vd. podrá encontrar programas donde las sentencias de **DATA** están al final del mismo, y en otros al principio.

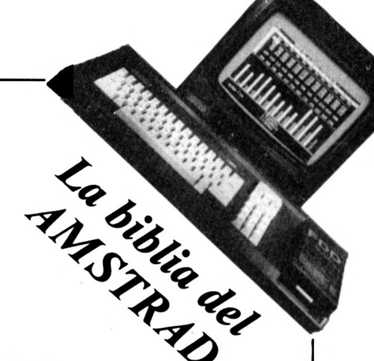
Dado que hemos aprendido el uso del bucle **FOR-NEXT**, ahora veremos qué pasa cuando se colocan sentencias de **DATA** en medio de un bucle, para eso borraremos la memoria y escribiremos el siguiente programa:

```
NEW
10 DATA 1, 2, 3, 4, 5
20 FOR N = 1 TO 5
30 READ A
40 PRINT A;
50 NEXT N
RUN
```

En este programa,la línea de **DATA** se encuentra fuera del bucle, así pues para ver lo que pasa, la moveremos dentro del mismo escribiéndola como línea 25 y ejecutándolo otra vez.

Habrá notado que no hay diferencia entre una ejecución y la otra, dado que como hemos pasado la variable N cinco veces por el bucle, y leído (**READ**) la letra A y a su vez impresa dicha letra con la sentencia **PRINT**, la letra A tendrá un valor diferente cada vez, dado que en cada paso se ha leído una parte de la sentencia DATA.

En algunos casos será necesario el tener que leer la misma DATA más de una vez sin tener que ejecutar (**RUN**) de



nuevo el programa, para este caso hay disponible una sentencia llamada **RESTORE** donde su función es que cuando un programa se encuentra con esta sentencia, todas las líneas de **DATA** son restauradas a su condición original aunque se hubiesen leído o no, estando listas para volver a ser leídas. Para demostrarlo, introduciremos la sentencia **RESTORE** en nuestro programa:

```
35 RESTORE
RUN
```

Sorpresa!!!! en la pantalla aparecen cinco unos en vez de 1 2 3 4 5, ¿puede Vd. descifrarlo?

En la línea 30 se lee la letra A como 1, inmediatamente en la línea 35 hacemos un **RESTORE** a la línea de **DATA** a su estado original, haciendo que el bucle **FOR-NEXT** lea (**READ**) la **DATA** cinco veces.

Las sentencias **DATA** y **READ** son muy comunes en los programas, cosa que no lo es la sentencia **RESTORE**.

Cadenas Variables

Hasta ahora hemos estado usando las letras de la A a la Z como Variables Numéricas, y podemos usar las mismas añadiendo el signo «$» a continuación de la letra (A$) para crear Cadenas Variables. Este tipo de cadenas se les puede asignar para indicar letras, palabras y/o combinaciones de letras, números y espacios. Escriba el siguiente programa:

```
NEW
10 INPUT "CUAL ES SU NOMBRE"; A$
20 PRINT "QUE TAL ESTA, "; A$
RUN
```

Como puede verse la palabra que hemos creado con su nombre, aparece en la ejecución del programa, así pues hemos aprendido dos formas de imprimir palabras, la primera tratada en los primeros Capítulos con la sentencia **PRINT, y la segunda a través de la sentencia INPUT** con cadenas variables.



Cambie el programa para que se lea:

```
10 READ A$
20 DATA COMPUTADOR AMSTRAD
30 PRINT "VEA MI "; A$
RUN

VEA MI COMPUTADOR AMSTRAD
```

Ahora vamos a usar dos Cadenas Variables para conseguir el mismo propósito, para eso modificaremos el programa:

```
10 READ A$
15 READ B$
20 DATA COMPUTADOR, AMSTRAD
30 PRINT "VEA MI "; A$; " "; B$
RUN
```

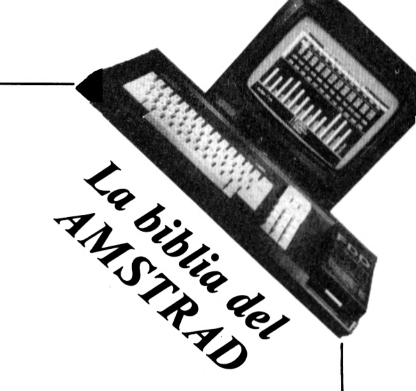
Analizando el programa tenemos lo siguiente:

En la línea 20 tenemos dos palabras de **DATA** separadas por una coma

La línea 10 lee (**READ**) la primera

La línea 15 lee (**READ**) la segunda

En la línea 30 hay cuatro expresiones de impresión, la primera imprime VEA MI dejando un espacio para la cadena variable, la segunda imprime el contenido en **A$**, COMPUNTADOR, la tercera deja un espacio correspondiente al que está entre comillas, y por último imprime la variable **B$**, AMSTRAD.



CAPITULO 20

BASIC INTERMEDIO

Hasta este momento, hemos estado tratando con un basic Elemental, así pues lo que viene a continuación será un poco más profundo y deberá prestar mucha atención.

Líneas de sentencias múltiples

EL BASIC permite el poner en una misma línea varias sentencias separadas por dos puntos (:), para así ahorrar espacio y hacer que el programa sea más corto. Por ejemplo:

```
100 FOR N = 1 TO 500
110 NEXT N
```

Se puede poner como sigue:

```
100 FOR N = 1 TO 500 : NEXT N
```

Este tipo de sentencias múltiples hay que entenderlas muy bien, dado que en algunos momentos son muy críticas como en el caso de **IF-THEN**.

Entre este programa:




```
NEW
10 INPUT "ENTRE UN NUMERO"; X
20 IF X = 3 THEN 50 : GOTO 70
30 PRINT "COMO HA LLEGADO HASTA AQUI?"
40 END
50 PRINT "X = 3"
60 END
70 PRINT "NO PUEDO IR DE AQUI PARA ALLA."
RUN
```

Ejecútelo varias veces con diferentes valores para que vea los resultados.

La línea 20 tiene un error lógico. Si la comprobación de la primera sentencia en la línea pasa, el control es mandado a la línea 50, pero si no pasa el control va a la siguiente línea 30, sin poder ser ejecutada la sentencia **GOTO 70** de la línea 20.

Nunca se podrá mandar el control en un línea con sentencias múltiples excepto en la primera sentencia. Si se fija en la línea 20, vera que hay forma de direccionar la porción de **GOTO** 70, dado que comparte el mismo número de línea que la primera sentencia en la misma línea.

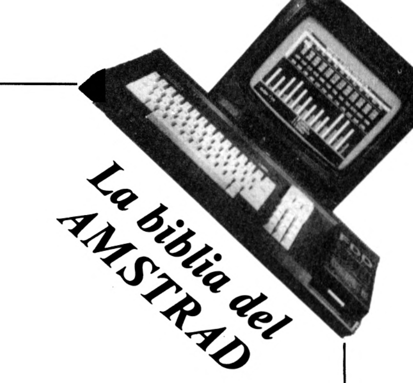
Nuevas Cadenas Variables

Hasta ahora hemos usado las cadenas variables **A$** y **B$**, pudiendo usarse todas las letras del alfabeto más los números del 0 al 9 y además combinacionaciones de dos letras, como por ejemplo **X$, DB$, PI$, B4$** etc.

Hay varios trucos para escribir menos y no cansarse cuando hay un programa largo que introducir en la máquina.

El primero es el usar el simbolo «?» en sustitución de la palabra PRINT, así pues lo probaremos escribiendo:

```
NEW
10 ? "SU NOMBRE"
LIST
```



y aparecerá en pantalla

```
10 PRINT "SU NOMBRE"
```

Pruebe ahora con:

? 3 * 4 y obtendrá

```
12
```

Otra abreviación muy común que se puede usar, es la que sustituye a la sentencia **REM** que se usará el símbolo «'», que aparecerá en muchos programas como:

```
50 X = Z * C / 4 + 33    'LA ECUACION SECRETA
```

El único sitio donde esta abreviatura no se puede usar es en las líneas donde se encuentre la sentencia **DATA**, pudiéndose resolver usando dos puntos (:) al final de la sentencia **DATA**. Para observar este fenómeno, vea el siguiente programa:

```
10 REM * PROGRAMA DE DEMOSTRACION *
20                              ' H = POSICION HORIZONTAL
30                              ' V = POSICION VERTICAL
40 CLS                          ' N = NUMERO DE BLOQUES VERTICALES
50 READ H, V, N : V1 = V        ' LEER DATA Y ALMACENARLA
60 IF N = 0 GOTO 60             ' BUCLE CUANDO NO HAY DATA
70 FOR H = H TO H + 2           ' 3 PASES PARA BLOQUE TRIPLE
80 FOR V = V TO V + N - 1       ' CONTADOR DE IMPRESION DE N BLOQUE
90 ? H, V: NEXT V               ' TERMINACION BUCLE HE IMPRESION
100 V = V1: NEXT H: GOTO 50     ' RESETEA V A LA LINEA DATA
1000 DATA 102,3,9,105,10,1,108,7,3,111,6,1 :' DATOS
1010 DATA 114,7,3,117,10,1,120,3,9,0,0,0   :' DATOS
```

Liste este programa para ver los resultados y así en el futuro usar estas abreviaciones.

PRINT??

Cuando se usan varias entradas en una sentencia de INPUT en una misma línea, y si nos olvidamos de separar cada entrada con una coma, aparecerá en pantalla, y en la siguiente línea el símbolo especial de ?? indicando que se ha olvidado de entrar más datos. Por ejemplo, practique con la siguiente línea.




```
NEW
10 INPUT A, B, C
RUN
```

NEXT opcional

Los bucles **FOR-NEXT** no siempre se especifica que **FOR** es para el **NEXT**, siendo muy útil cuando los bucles están entrelazados. Entre el siguiente programa:

```
NEW
10 FOR N = 1 TO 5 : PRINT N
20 FOR Q = 1 TO 3 : PRINT ,Q
30 FOR R = 1 TO 4 : PRINT ,,R
40 NEXT: NEXT: NEXT
RUN
```

Ejecútelo varias veces para tomar buen contacto con este tipo de bucle y así poder evaluarlo.

Este método no se podrá usar en programas donde se encuentren bucles de prueba donde los cuales se deberán partir. Otra forma de abreviatura sería como ésta:

```
40 NEXT R, Q, N
```

La sentencia IF-THEN-ELSE

La palabra **ELSE** es un tipo de condición muy interesante para sentencias de ramificación, permitiendo que si una comprobación es falsa no pase a la siguiente línea. Pruebe con el siguiente programa:

```
NEW
1 CLS : PRINT
10 INPUT "ENTRAR UN NUMERO"; N
20 IF N = 0 PRINT "CERO" ELSE PRINT "NO ES CERO"
30 PRINT: LIST
RUN
```

La sentencia POS (N)

Esta sentencia proporciona una posición horizontal del cursor que ocupa en ese momento correspondiente a un cauce de salida dado. Entre este simple programa:




```
NEW
1 CLS : PRINT
10 INPUT "UN NUMERO ENTRE -10 Y 53"; A
20 PRINT TAB(10 + A)
30 PRINT POS (N)
40 PRINT "EL NUMERO DE LA SIGUIENTE PASICION DE PRINT"
90 PRINT: LIST
RUN
```

En la línea 30 nos encontramos con la sentencia **POS** y la letra **N** que se encuentra entre paréntesis es de pura ficción. Esta sentencia informará cualquier posición del cursor hasta la 63.



CAPITULO 21

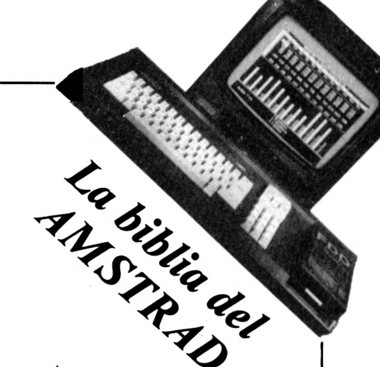
EL CONJUNTO ASC II

La misión de este Capítulo es el aprender cómo usa **ASC** y **CHR$**, y antes de empezar se deberá ver lo que se llama «Conjunto ASC II» que su significado es «American Standard Code for Information Interchange». Dado que el Computador solamente almacena números y no letras o símbolos, es importante el informar al sistema las equivalencias de esas letras y símbolos, y para ello existe una tabla llamada **ASC II**, la cual la podrá encontrar en su manual de usuario y también en este libro.

Teclee este programa:

```
NEW
10 FOR N = 32 TO 255
20 PRINT "NUMERO ASCII"; N;
30 PRINT "EQUIVALENTE A", CHR$(N)
40 FOR T = 1 TO 500: NEXT T
50 NEXT N
RUN
```

Este programa imprimirá en nuestra pantalla los códigos **ASCII** desde el número 32 y el 255 estando incluidos



todas las letras tanto mayúsculas como minúsculas, así como símbolos y gráficos.

Que es un CHR$ (N) ??

Hasta ahora hemos estado usando la función **CHR$** sin saber realmente lo que significaba, aunque nos dábamos una idea. La misión de la función **CHR$ (N)** es producir un carácter **ASC II** (o una acción de control) especificada por el codigo N. Es una forma de convertir del codigo **ASC II** a un carácter **ASC II** y permitirnos el poder jugar con los caracteres igual que lo podríamos hacer con números.

Entrar este simple programa para demostrar la acción del **CHR$ (N)**:

```
NEW
10 INPUT "PONGA UN NUMERO ENTRE 33 Y 127"; N
20 PRINT CHR$(N)
30 RUN
RUN
```

Naturalmente con este programa hemos visto los diferentes números que corresponden a los diferentes caracteres, pero hay más números dentro de esta tabla **ASC II**, los cuales sirven de control, por lo que le sugiero que pruebe con los números de control en el programa anterior, y verá los resultados. Para más orientación, use la tabla **ASC II** que se encuentra en su manual de usuario.

Que es unASC ($)??

La función ASC es exactamente lo opuesto a **CHR$ (N)**, convirtiendo de alguna forma los caracteres **ASC II** en su correspondiente número **ASC II**.

Entre este programa para demostrárselo:

```
NEW
10 INPUT "ENTRE CUALQUIER LETRA, NUMERO O CARACTER"; A$
20 PRINT "SU NUMERO ASCII ES:"; ASC(A$)
30 PRINT
40 GOTO 10
RUN
```



Como podrá comprobar, este programa le proporciona cualquier número **ASCII** que Vd. quiera saber de cualquier carácter.

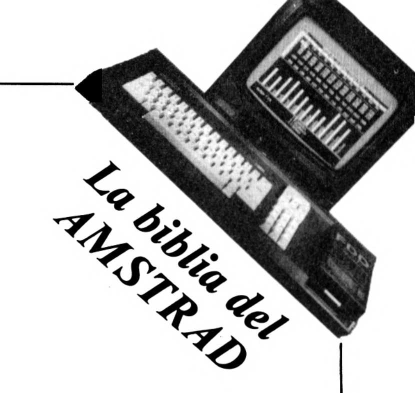
Una segunda forma de usar la función **ASC** sería de la siguiente forma:

```
10 PRINT ASC("A")
```

Con lo cual cualquier letra que esté dentro del paréntesis y entre comillas, le dará su resultado en numeración **ASCII**.

Antes de que podamos entender lo que hacemos, deberemos aprender un poco más sobre las cadenas, las cuales están ligadas de alguna forma al código **ASCII**.

(Continuará)



CAPITULO 22

GENERALIZACION DE CADENAS

Uno de los más importantes aspectos del Manejador de Cadenas, es la habilidad de la comparación de las mismas. Hasta ahora hemos podido comparar variables numéricas, pero ¿cómo comparar cadenas de letras o palabras? Muy sencillo, habrá visto Vd. que en el Capítulo anterior hemos tratado sobre el código **ASCII** con lo cual nos da paso para poder demostrar que la comparación de Cadenas de letras o palabras se puede efectuar, dado que el Computador compara el código **ASCII** de esas letras o palabras, en otras palabras, traduce las letras a números.

Entre el siguiente programa:

```
NEW
1 CLS
10 INPUT "ESCRIBA SU NOMBRE"; A$
20 IF A$ = "AEIOUSSDTA" THEN 50
30 PRINT "LO SIENTO SU NOMBRE NO ES AEIOUSSDTA!"
40 END
50 PRINT "VD. NO SABE DELETREAR SU NOMBRE?"
RUN
```



Durante el proceso de comparar el nombre que Vd. ha entrado en la línea 10 como A$, el código ASCII traduce letra a letra la palabra, para más tarde compararla con la que hemos puesto en la línea 20 que a su vez ha sido traducida por el mismo código.

Si se lee (READ) una cadena que no tiene, comas, punto y coma, o espacios, no será necesario el ponerla entre comillas, aunque si la pone no pasará nada pero es muy pesado.

EJERCICIO 22-1

Escriba un programa donde se comparen dos cadenas entradas desde el teclado, e imprímalas en orden alfabético.

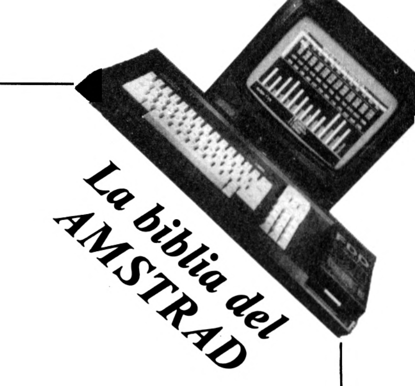
Escriba el siguiente programa donde se leerá cadenas de datos de una línea de datos:

```
NEW
1 CLS
10 READ A$, B$, C$
20 PRINT A$
30 PRINR B$
40 PRINT C$
100 DATA AMSTRAD COMPUTER, LOCOMOTIVE SOFTWARE, INGLATERRA, 12345
RUN
```

Vea los resultados con muchas atención, que serán los siguientes:

```
AMSTRAD COMPUTER
LOCOMOTIVE SOFTWARE
INGLATERRA
```

Ha notado que no ha salido en pantalla la numeración de 12345, y eso es dado porque en la sentencia **READ** solamente tenemos tres conceptos, y en la de **DATA** hemos puesto cuatro. Así pues para poder imprimir la coma como parte de la cadena, habrá que poner entre comillas la sen-



tencia que queramos. Modifiquemos la línea 100 para ver los resultados:

```
100  DATA AMSTRAD COMPUTER, "LOCOMOTIVE SOFTWARE, INGLATERRA", 12345
RUN
```

Ahora verá la diferencia. Pero, ¿que pasaría si pusiéramos toda la línea de **DATA** entre comillas dejando la que está? Modifique la línea 100 otra vez:

```
100 DATA "AMSTRAD COMPUTER, "LOCOMOTIVE SOFTWARE, INGLATERRA", 12345"
RUN
```

Un desastre, tenemos un **ERROR** de sintaxis dado que el Computador no es lo suficientemente inteligente para descifrar cómo dividir las comillas dentro de una cadena constante.



CAPITULO 23

MEDICION DE LAS CADENAS

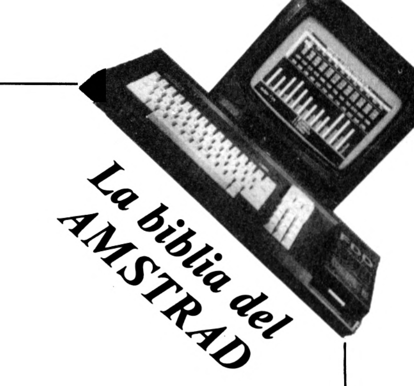
Una de las necesidades más frecuentes en programación, es el saber la longitud de una Cadena, con lo que usando la función **LEN** se puede resolver el problema.

Entre el siguiente programa:

```
NEW
1 CLS: PRINT
10 INPUT "ENTRE UNA CADENA DE CARACTERES"; A$
20 L = LEN (A$)
30 PRINT A$; "TIENE"; L ;"CARACTERES"
90 PRINT: LIST
```

Ejecute este programa varias veces, poniendo su nombre y otras combinaciones de letras y números. No ponga ninguna coma entre los nombres o números, pero sí puede poner comillas.

El comando **LEN** tiene una variación muy significante, y es que no es práctico a no ser que realmente se necesite, y para demostrarlo, cambiaremos las líneas 10, 20, y 30 de nuestro programa:




```
10 INPUT "ENTRE UN NUMERO"; A
20 L = LEN(A)
30 PRINT A; "TIENE"; L ;"CARACTERES"
RUN
```

Veremos que nos da un **ERROR**, dado que tratamos de introducir un número en una función **LEN**, donde se requiere una cadena. Ahora variaremos la línea 20 cambiando la función **LEN** para que sea una cadena:

```
20 L = LEN("A")
RUN
```

Parece que no importa qué número Vd. meta con la sentencia **INPUT** dado que siempre la respuesta es 1 carácter. La respuesta a este fenómeno es muy sencilla, dado que la función **LEN** evalúa la longitud de lo que está entre paréntesis (o comillas). Al principio toma la cadena y mide su longitud, pues como estábamos midiendo lo que estaba entre comillas, la longitud no cambia con el valor de A. Como dijimos anteriormente, la función **LEN** tiene sus limitaciones, pero nos dice la longitud de lo que hay.

Cambie el programa a su estado original como aparece al principio del Capítulo.

DEFSTR

El comando **DEFSTR** nos permite definir qué variable debe ser una cadena variable, y así no tener que usar el signo $. Ponga esta línea en el programa que está residente en memoria:

```
5 DEFSTR A
```

Usando el Editor, quite los signos $ de las líneas 10, 20, y 30, y ejecute el programa.

Comprobará que funciona perfectamente, dado que hemos declarado en la línea 5 una cadena variable. Ahora cambiemos otra vez la línea 5 de la siguiente forma:

```
5 DEFSTR A-Z
RUN
```



En este caso hemos creado un **ERROR** dado que la L de la línea 20 es ahora también una cadena, y dado que la función **LEN** nos da la longitud de una cadena, como número, esto no funciona en este caso con L.

El comando **DEFSTR** puede usarse para definir variables individuales, por ejemplo:

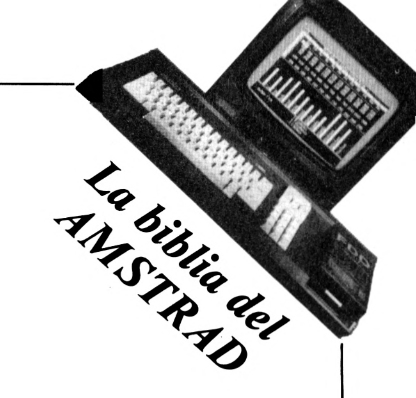
```
DEFSTR A, N, Z
```

EJERCICIO 23-1

Escriba un programa donde se use la función LEN para comprobar la longitud de una cadena entrada desde el teclado, e imprima un mensaje donde nos informe que la cadena tiene más de 10 carácteres.

EJERCICIO 23-2

Escriba un programa donde si se entra una palabra desde el teclado la compare con un palabra secreta (Password), y si coinciden, que saque un mensaje que diga PALABRA SECRETA CORRECTA, TIENE ACCESO, pero si no que dé el siguiente mensaje PALABRA INCORRECTA, LO SIENTO. Almacene el número **ASCII** de cada letra de la palabra secreta en la línea de **DATA**. Lea (**READ**) cada valor y use **CHR$** para crear la cadena de la palabra secreta.



CAPITULO 24

LAS FUNCIONES VAL ($) Y STR$ (N)

Por definición, si nosotros convertimos una variable numérica (que puede almacenar cualquier número) a una cadena variable (que puede almacenar cualquier cosa), el contenido de la nueva cadena seguirá siendo el número original. Consecuentemente si cambiamos una cadena variable a una variable numérica, no podremos cambiar ninguna letra u otros caracteres a números, solamente los números en una cadena podrán ser convertidos a una variable numérica (no se confunda con una conversión en **ASCII**).

VAL

La función **VAL** convierte una cadena variable conteniendo un número a numérica, si el número está al comienzo de la cadena. Probaremos este programa con la función **VAL**:

```
NEW
1 CLS: PRINT
10 INPUT "QUE CADENA CONVERTIRE A UN NUMERO"; A$
20 A = VAL(A$)
```




```
30 PRINT "EL VALOR NUMERICO DE "; A$ ;" ES "; A
90 PRINT: GOTO 10
RUN
```

Entre diferentes bloques como:

```
12345
ASDF
123ASD
1,2,3
A,B,C
```

A continuación quitaremos de las líneas 10, 20 y 30 el signo de $ y lo ejecutaremos de nuevo entrando los mismos números y letras.

El resultado es nefasto, dado que la función **VAL** sólo evalúa cadenas y nosotros hemos puesto a la variable A como valor numérico.

STR$

Ahora trataremos con lo opuesto anteriormente, y es el convertir una variable numérica en una cadena variable. Cambie el programa anterior:

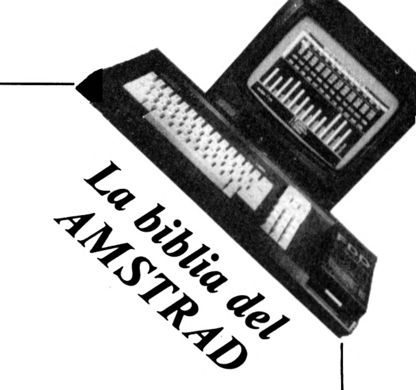
```
NEW
1 CLS: PRINT
10 INPUT "QUE NUMERO DESEO CONVERTIR A CADENA"; A
20 A$ = STR$(A)
30 PRINT "EL VALOR DE LA CADENA DE "; A ;"ES"; A$
90 PRINT: GOTO 10
RUN
```

Use las mismas entradas que utilizamos en el programa de **VAL**.

Aunque este Capítulo ha sido muy corto, no deja de ser muy interesante y sería importante que Vd. practicara con él para que en un futuro pueda usar estas dos funciones.

EJERCICIO 24-1

Escriba un programa donde se le preguntará la calle y



el número donde vive, y usando la función **VAL** extraer el número de la calle. Sumar el número 4 al número de la calle y reportar este nuevo número como el de su vecino.

EJERCICIO 24-2

Escriba un programa usando la función **STR$** donde se impriman 20 números de stock de artículos: 101WT, 102WT, 103W,.....120WT.



CAPITULO 25

Tres diferentes funciones pero muy similares son usadas para poder jugar con las cadenas, éstas son **LEFT$, RIGHT$ Y MID$**. Empezaremos nuestro ejercicio con este programa:

```
NEW
1 CLS: PRINT
30 S$ = "JOSE ESTUDIO AQUI"
60 PRINT LEFT$(S$,4),
70 PRINT MID$(S$,8,7),
80 PRINT RIGHT$(S$,4)
90 PRINT: LIST
RUN
```

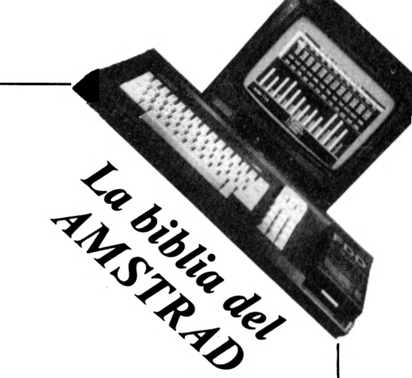
En la pantalla aparecerá:

```
JOSE          ESTUDIO          AQUI
```

El aprender el uso de estas funciones es muy simple, y estudiando el programa muy despacio y con atención mientras se le explica lo que pasa.

**LEFT$** imprime los 4 carácteres más significativos de la izquierda en la cadena llamada S$.

**MID$** imprime los 7 caracteres en la cadena llamada S$ empezando por el octavo carácter a la izquierda.



**RIGHT$** imprime los 4 caracteres más significativos a la derecha en la cadena llamada S$.

Hemos añadido comas al final de las líneas 60 y 70 en orden para que todo se imprima en una sola línea. Moveremos algunas líneas en el programa, para ejercitar con estas funciones.

Primero moveremos la línea 70 a la 50:

```
50 PRINT MID$(S$,8,7),
RUN
```

Se obtendrá lo siguiente:

```
ESTUDIO          JOSE       AQUI
```

Ahora moveremos la línea 80 a la 40 añadiendo una coma al final:

```
40 PRINT RIGHT$(S$,4)
```

Obteniendo lo siguiente:

```
AQUI         ESTUDIO       JOSE
```

Estas tres funciones pueden hacer maravillas con las cadenas, por lo que entraremos un nuevo programa para examinarlas con más detalle:

```
NEW
1 CLS: PRINT
10 FOR N = 1 TO 17
20 PRINT "N = "; N,
30 S$ = "JOSE ESTUDIO AQUI"
40 PRINT LEFT$(S$,N)
50 FOR T = 1 TO 500: NEXT T
60 NEXT N
90 PRINT: LIST
RUN
```

Una imagen relentizada lo explicará más deprisa que muchas palabras. La función **LEFT$** toma N letras de la parte izquierda de la cadena y las imprime. Vea cómo esto puede usarse para partir los tres primeros dígitos de un



número telefónico, o la primera letra de un nombre cuando se hace una selección.

Cambie la línea 10

```
10 FOR N = 1 TO 20
RUN
```

Como podemos ver, aunque solamente hay 17 caracteres en esta cadena, los remanentes los ignora. Cambie la línea 10 a su estado original.

La función **RIGHT$** trabaja igual que la anterior, pero desde la derecha, para demostrarlo cambiaremos la línea 40:

```
40 PRINT RIGHT$(S$,N)
RUN
```

Ahora ejercitaremos con la función **MID$** cambiando la línea 40:

```
40 PRINT MID$(S$,N,1)
RUN
```

Metódicamente borre la cadena de izquierda a derecha tomando una letra a su tiempo. Aquí lo hemos relentizado con un bucle de retardo en la línea 50 para que se pueda ver mejor su ejecución.

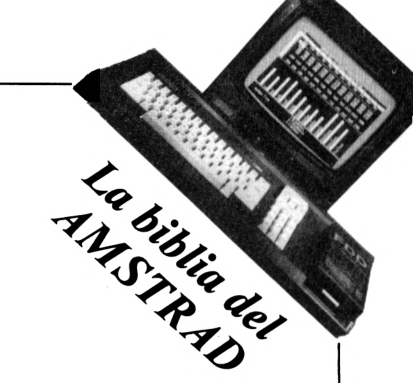
Con solamente un cambio mínimo, podemos hacer que la función **MID$** actúe como la función **LEFT$** haciendo un cambio en la línea 40:

```
40 PRINT MID$(S$,1,N)
RUN
```

Se imprimen N caracteres empezando por el número uno a la izquierda.

Esta función **MID$** también puede simular a la función **RIGHT$**, cambiando la línea 40:

```
40 PRINT MID$(S$,18-N,N)
RUN
```



Ahora seleccionaremos una posición específica en la cadena imprimiendo su carácter, y para eso deberemos entrar el siguiente programa:

```
NEW
1 CLS: PRINT
10 INPUT "QUE CARACTER # DESEA IMPRIMIR"; N
30 S$ = "JOSE ESTUDIO AQUI"
40 PRINT MID$(S$,N,1)
50 FOR T = 1 TO 500: NEXT T
90 PRINT: LIST
RUN
```

Y para terminar con esta sección, asignaremos a una de estas sentencias una variable, y ésta puede usarse como comprobación de las otras. Cambie lo siguiente:

```
40 V$ = MID$(S$,N,1)
45 PRINT V$
```

EJERCICIO 25-1

Escriba un programa donde se haga una pregunta como «ESTE ES UN ORDENADOR INTELIGENTE», y entrando como contestación SI o NO. Si el primer carácter de la respuesta es S, deberá imprimir AFIRMATIVO, pero si el primer carácter de la respuesta es N deberá imprimirse NEGATIVO. En el caso que el primer carácter no fuese S o N deberá imprimirse ESTO NO ES UNA PREGUNTA DE SI O NO y devolver el control a la sentencia **INPUT**.

EJERCICIO 25-2

Escriba un programa donde se lean (**READ**) los siguientes números de piezas: N106WT, A208FM y Z154DX, usando la función **MID$** para encontrar los números y almacenarlos en una cadena (caracteres del 2-4), reportando el número de pieza con el valor numérico mayor.



CAPITULO 26

Funciones de Pantalla

La pantalla de este Computador es muy parecida a una retícula de malla cuadrada con puntos de imagen (Pixels) que se pueden ejecutar por separado. Cada uno consta de un cierto número de bits en un área de memoria, y todo depende del modo en que se programe la pantalla.

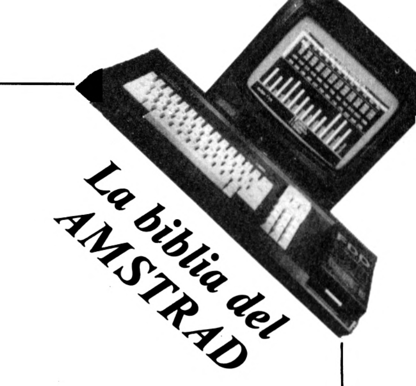
Hay dos formatos de pantalla, una de Textos, y otra de gráficos. Para la primera, ésta se divide en un cierto número de carácteres o símbolos imprimiéndolos en la pantalla. Para el formato de gráficos se especifica individualmente cada punto de imagen incluyendo la facilidad para dibujar líneas así como determinar ciertos colores.

Comandos y funciones

BORDER: Fija el color del borde de la pantalla usando la tabla de colores que viene en su manual.

CLG: Su misión es de borrar la pantalla en modo gráfico.

CzlS: Borra la pantalla en modo de texto.



DRAW: Su misión es la de trazar una línea recta en el modo de gráficos hasta una posició n absoluta.

DRAWR: Su misión es la de trazar una línea recta en el modo de gráficos hasta una posición relativa.

INK: Fija el color de la tinta.

LOCATE: Coloca al cursor en una posición con las dos coordenadas (vertical y horizontal).

MODE: Modifica el modo de pantalla. Existen tres modos de pantalla:

MODE 0 que corresponde a 25 líneas de 20 caracteres cada una, o 160 puntos de ancho por 200 de altura, así como 16 tintas posibles (4 bits por pixel).

MODE 1 corresponde a 25 líneas de 40 caracteres cada una, o 320 punto de ancho por 200 de altura, así como 4 tintas disponibles (2 bits por pixel).

MODE 2 fija 25 líneas de 80 carácteres cada una, o 640 puntos de ancho por 200 de altura, así como 2 tintas disponibles (1 bit por pixel).

MOVE: Mueve el cursor en modo gráfico hasta una posición absoluta.

MOVER: Mueve el cursor en modo gráfico hasta una posición relativa.

ORIGIN: Coloca el origen de las coordenadas para modo de gráficos.

PAPER: Coloca el color del papel o fondo de pantalla.

PEN: Coloca el color de la pluma o color de los caracteres.

PLOT: Dibuja un punto en pantalla en modo de gráficos en una posición absoluta.

PLOTR: Dibuja un punto en pantalla en modo de gráficos en una posición relativa.



POS: Posiciona horizontalmente el cursor en modo de textos.

SPEED INK: Coloca la velocidad de parpadeo de las tintas.

SYMBOL: Coloca la matriz de los caracteres definidos por el usuario.

SYMBOL AFTER: Coloca como caracteres definibles por el usuario los de después del dado.

TAG: Fija el modo de texto en el cursor de gráficos.

TAGOFF: Quita el modo de texto en el cursor de gráficos.

TEST: Número de tinta usada para un gráfico con coordenadas absolutas.

TESTR: Número de tinta usada para un punto en modo gráfico con coordenadas relativas.

VPOS: Posición vertical del cursor de textos.

WINDOW: Coloca la pantalla para una proyección de texto.

WINDOW SWAP: Cambia la pantalla correspondiente a los dos cauces.

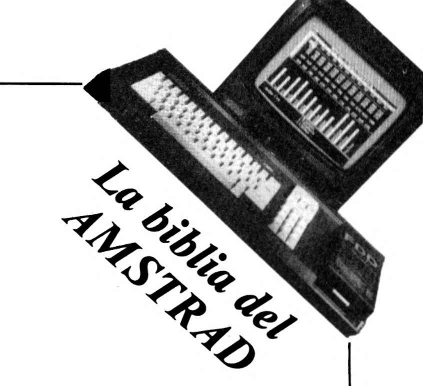
XPOS: Posiciona horizontalmente el cursor de gráficos.

YPOS: Posiciona verticalmente el cursor de gráficos.

El color con que aparecen los pixels (puntos) de imagen por los comandos **PLOT, DRAW o TAG, depende del modo de tinta para gráficos que pueden ser uno de los siguientes:**

NORMAL: Color de tinta nueva.
XOR: Color O-lógico exclusivo entre las tintas nuevas y viejas.

AND: Color Y-Lógico entre las tintas nuevas y viejas.
OR: Color O-lógico entre las tintas nuevas y viejas.



Dado que no se puede meter en este Capítulo todos los ejemplos que desearíamos mostrarle, con relación a estos comandos, solamente les orientaremos con estos dos programas, la capacidad del manejador de pantalla y los efectos que se pueden conseguir.

En el primer programa, se ilustrará cómo los comandos de **PEN** y **PAPER** se pueden usar para especificar colores específicos usados en diferentes ventanas o pantallas. Siempre que Vd. quiera usar diferentes colores para los comandos de **PEN** y **PAPER**, comprobar los colores primero para ver si son compatibles.

A continuación escribiremos este programa para demostrar este efecto, y así Vd. podrá modificarlo para comprobar su efecto en pantalla.

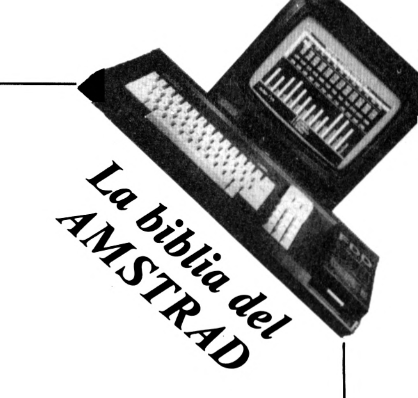
```
NEW
10 BORDER 0
20 WINDOW#1,1,40,3,5
30 WINDOW#2,1,40,7,10
40 WINDOW#3,1,19,11,24
50 WINDOW#4,20,20,11,24
60 A$="ESTO ES UNA PRUEBA PARA VER EL EFECTO DE LA VENTANA."
70 PAPER#1,0
80 PAPER#2,1
90 PAPER#3,2
100 PAPER#4,3
110 PEN#1,3
120 PEN#2,2
130 PEN#3,1
140 PEN#4,0
150 CLS
160 FOR N = 1 TO 4
170 PRINT # N, A$: NEXT
RUN
```

El otro programa, se refiere a los efectos que se pueden producir para falsear una palabra, o conjunto de palabras, y así poder llamar la atención de alguna cosa. Este tipo de rutina, le será muy práctico para que Vd. pueda impresonionar a los amigos cuando desarrolle sus programas

```
NEW
10 CLS
20 INK 2,6,8
```




```
30 INK 3,12,19
40 PEN 2
50 PRINT: PRINT "MUCHA ATENCION!!!"
60 PRINT:PRINT:PRINT
70 PEN 3
80 PRINT "MATERIAL PELIGROSO!!!"
90 PEN 1
RUN
```



CAPITULO 27

GRÁFICOS

Los gráficos de este Computador son uno de los mejores, en comparación con otros equipos. En este Capítulo cubriremos los diferentes aspectos de creación de gráficos sin usar la sentencia **PRINT** ni la función **CHR$**.

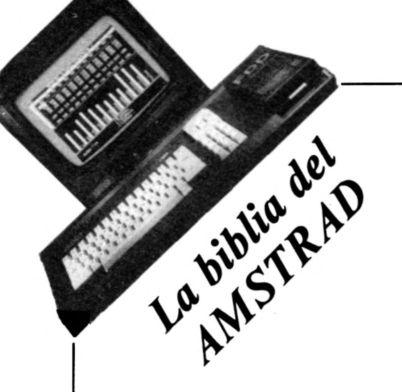
Los puntos con que el Computador trabaja se les llaman **PIXELS**, y cada uno de ellos que el Computador controla, corresponden al tamaño de varios puntos de un Televisor normal.

En el modo de gráficos, todos, los comandos hacen uso de las coordinadas del sistema X e Y, las cuales también se usan con el comando **LOCATE**, que sirve para colocar el cursor en un sitio de la zona de pantalla comenzando en la esquina superior izquierda, que corresponden a la coordinada X (eje horizontal) e I (eje vertical):

LOCATE X, Y

Para demostrar su acción, escribiremos un programa, ejecutándolo y viendo su resultado:

```
NEW
10 CLS
```




```
20 MODE 1
30 LOCATE 20,12
40 PRINT CHR$(240)
RUN
```

Una vez ejecutado, verá que aparece un símbolo gráfico en el centro de la pantalla, correspondiendo la numeración de este comando a los siguiente:

El número 20 de **LOCATE** es la coordenada horizontal (X) en la gama de 1 a 40.

El número 12 de **LOCATE** es la coordenada vertical (Y) en la gama de 1 a 25.

El número 240 de la función **CHR$** corresponde al símbolo que aparece en la pantalla. En vez de usar **PRINT CHR$**, Vd. puede poner una frase en cualquier sitio dela pantalla usando **PRINT**». He de aclarar, que esta función de **LOCATE** no se puede usar para formatear la salida por impresora de papel.

Hay que tener en cuenta cuando se marcan estas coordinadas, en qué modo de pantalla se encuentra.

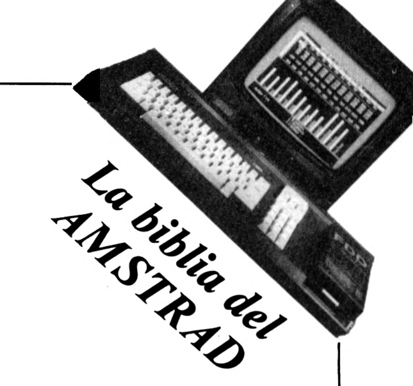
Para todos los modos de gráficos, el rango de X es de 0 a 399 siendo muy conveniente dado que si Vd. hace un programa de gráficos en un tipo de modo, y desea ejecutarlo en otro modo, no tendrá que revisar todo el programa cambiando la numeración.

Las siguientes normas, le ayudarán a dibujar un mapa de gráficos usando un papel gráfico:

1.— Tome un papel milimetrado tamaño A4 en cm o mm.

2.— Numere la coordenada X (horizontal) en saltos de 25, hasta 600, y la coordinada y (vertical), también en saltos de 25, hasta 400.

3.— Usando papel transparente (tipo cebolla), póngalo encima del papel milimetrado, dibujando el gráfico deseado y leyendo sus coordenadas.



Plotting

Un gráfico es una serie de puntos que en su conjunto se convierten en una información gráfica. Existen dos comandos para crear esta información, **PLOT** y **PLOTR**, diferenciándose entre ellos dado que el comando **PLOT** usa coordenadas absolutas, y **PLOTR** usa coordenadas relativas a la posición del cursor. Por ejemplo, si Vd. usa **PLOT** 0,0 el plot estará en el punto 0,0 o sea en la esquina inferior izquierda, pero si usa **PLOTR** 0,0 el plot estará donde esté el cursor en ese momento.

En el siguiente programa se demuestra cómo se pueden dibujar tres gráficos al mismo tiempo, a una velocidad razonable:

```
NEW
10 MODE 0
20 INK 0,0
30 FOR X = 1 TO 639
40 Y = 200 + 200 * SIN (2*PI*X/639)
50 PLOT X, Y, 1
60 Y = 200 + 200 * (SIN (2*PI*X/639)^ 2)
70 PLOT X, Y, 2
80 Y = 200 + 200 * (SIN (2*PI*X/639)^ 3)
90 PLOT X, Y, 3
100 NEXT
110 GOTO 110
RUN
```

Examinaremos las partes más esenciales del programa:

En la línea 30 se establece los diferentes valores para X, para que cubra toda la pantalla, y para cada valor de X, son calculados tres valores de Y en las líneas 40, 60 y 80. Cada valor se obtiene del seno del ángulo con un valor en radianes de X/639, el otro se obtiene del cuadrado del seno de ese ángulo, y el tercero se obtiene del cubo del seno del ángulo. Dicho seno tiene un valor que está entre –1 y +1, así pues como el valor de 1, en la dirección Y no significa nada, lo multiplicaremos por 200, por lo tanto el seno tendrá un valor entre –200 y +200. Dado que no podemos ha-



cer un plot de –200 en pantalla, añadiremos 200 a cada valor teniendo un valor que variará entre 0 y 400 para Y. El otro punto es que usamos como ángulo 2 * PI * X/639 para poder hacer mediciones en radianes. Estas funciones de ángulo tienen un rango de valores que van de cero radianes a 2 * PI radianes. Cuando X = 0, 2 * *PI * X/639 será cero radianes a la parte izquierda del gráfico, cuando X = 639 en la parte derecha del gráfico, el ángulo será 2 * PI * 639/639 que será 2 * PI radianes.

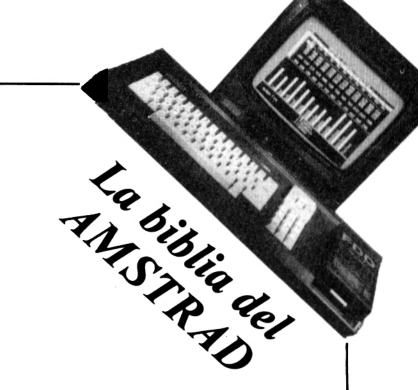
Una vez ejecutado este programa, podrá hacer pruebas cambiando la resolución de pantalla, modificando la línea 10 con los modos **MODE 1** y **MODE 2**, viendo así los resultados.

Dibujando líneas

Hay dos comandos para realizar esta función, **MOVE** y **DRAW**, el primero, como puede imaginárselo por su nombre, mueve el cursor de gráficos, pero como no puede verlo por ser invisible, Vd. no verá nada cuando este comando se ejecuta. El segundo dibuja la línea deseada, usando tanto el primer comando como éste, los mismos número de coordenadas X e Y que se usaron con el comando **PLOT**, además el comando **DRAW** usa un tercer número destinado a la tinta (**INKPOT**).

Escribamos un programa como ejemplo, para ver sus efectos en pantalla:

```
NEW
10 ON BREAK GOSUB 1000
20 MODE 0
30 MOVE 150,100
40 DRAW 150,300
50 DRAW 450,300
60 DRAW 450,100
70 DRAW 150,100
80 GOTO 80
1000 MODE 1
1010 LIST
RUN
```



Este programa no es nada elaborado, y lo único que hace es dibujar un cuadrado en la pantalla. La razón por la que no hemos puesto ningún número de tinta en los comandos de **DRAW**, es que el color que hemos usado es el mismo que había la última vez que se usaron los comandos **DRAW** o **PLOT**.

Como podrá Vd. comprobar, la instrucción de la línea 10, se ha puesto para que si quiere salirse del programa y listarlo, pulsando dos veces a la tecla **ESC**, el Computador pase a modo de pantalla **MODE 1** para facilitar su lectura, dado que en **MODE 0** es un poco difícil.

Otra variación de este programa es el siguiente:

```
NEW
10 ON BREAK GOSUB 1000
20 MODE 0
30 MOVE 150,100
40 DRAWR 0,200,2
50 DRAWR 300,0,2
60 DRAWR 0,-200
70 DRAWR -300,0
80 GOTO 80
1000 MODE 1
1010 LIST
RUN
```

En este programa se usa el comando **DRAWR** para dibujar un cuadrado usando además número de color.

Dibujando círculos

Dado que el dibujo consiste en líneas rectas más o menos largas, para algunos tipos de dibujos como, círculos, elipses y arcos se necesitarían otros comandos. Este Computador no tiene el comando de CIRCLE ni ningún otro para rellenar de color dicha figura, así pues se deberá usar los comandos de líneas. Para demostrar cómo se puede dibujar un círculo, escribiremos el siguiente programa:

```
NEW
10 MODE 1
```




```
20 ORIGIN 320,200: R = 100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1010 FOR N = 1 TO 360
1020 PLOT R * SIN (N), R * COS (N), 2
1030 NEXT
1040 RETURN
RUN
```

En la línea 20 se centra el círculo como origen, así como se le da un valor del radio R.

La línea 30 se llama a la subrutina para dibujar el círculo, siendo ésta la de la línea 1000 **DEG** que coloca los ángulos que han de usarse en unidades de grados.

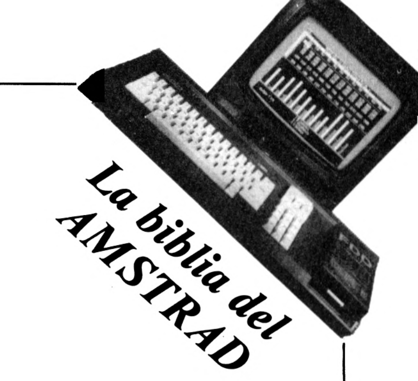
La línea 1010 da comienzo el bucle que permitirá el dibujar el círculo.

Este programa se ejecuta muy lentamente, así pues pondremos una alternativa para acelerar su ejecución, que será la siguiente:

```
NEW
10 MODE 1
20 ORIGIN 320,200: R = 100
30 GOSUB 1000
40 ORIGIN 0,0
50 GOTO 50
1000 DEG
1005 MOVER 0,R
1010 FOR N = 0 TO 360 STEP 10
1020 DRAW R * SIN (N), R * COS (N)
1030 NEXT
1040 RETURN
RUN
```

Animación con la tinta (INK)

Un método muy práctico de animación de un dibujo, es el usar el comando **INK**. Supongamos que lo que queremos animar es un dibujo que usa la función **CHR$**, pero se



usa los comandos **DRAW** y **MOVE**, naturalmente Vd. podrá dibujar una figura, borrarla, y volverla a dibujar en otro sitio. Un método más comodo es el usar el comando **INK**, con el cual si se le está cambiando su color al mismo que el fondo de la pantalla, el dibujo desaparecerá y aparecerá cada vez que cambiemos de color, así de simple.

Para demostrar este efecto de «tinta invisible», escribiremos el siguiente programa:

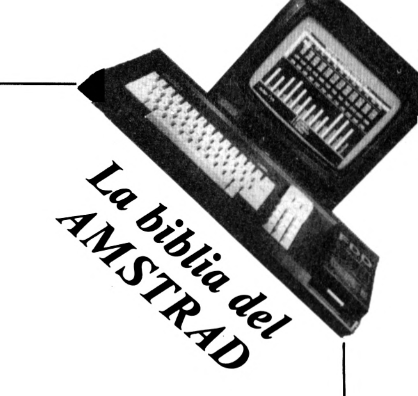
```
NEW
10 MODE 0
20 FOR CL = 2 TO 15
30 INK CL,1: NEXT
40 FOR NR = 1 TO 4
50 READ X,Y: MOVE X,Y
60 FOR LN = 1 TO 4
70 READ X,Y
80 DRAW X,Y,NR+1
90 NEXT: NEXT
100 WHILE INKEY (47) = -1
110 FOR NR = 2 TO 5
115 CALL &BD19
120 INK NR,24
130 FOR X = 1 TO 30: NEXT
140 INK NR,1
150 NEXT
160 WEND
170 DATA 300,300,350,300,350,100,300,100,300,300
180 DATA 385,288,415,252,270,113,235,148.385,288
190 DATA 225,225,425,225,425,175,225,175,225,225
200 DATA 230,255,275,293,410,148,377,110,230,255
210 MODE 1: END
RUN
```

La primera parte de este programa, pone los colores de 2 a 15 al color del fondo de la pantalla que es el número 1. Las líneas de la 40 a la 90 dibujan cuatro rectángulos, siendo uno vertical y los otros tres desplazados 45 grados con relación al primero. El bucle comienza en la línea 100 continuando hasta que Vd. presione la barra de espacio. Hemos introducido la línea 115 con la función **CALL &BD19**



para que el dibujo se ejecute mejor, pero si quiere ver el efecto de esta función, quite esta línea y verá lo que pasa.

En este Capítulo, realmente no se ha cubierto todas las posibilidades que tiene este Computador en sus funciones de gráficos, pero hemos cubierto algunos aspectos muy interesantes.



CAPITULO 28

FUNCIONES MATEMATICAS

En algunos casos donde los números son enteros (entre –32768 y +32767), la ejecución puede ser aumentada declarándolos ser números enteros usando el signo % o la sentencia **DEFINT**. Entre el siguiente programa:

```
NEW
20 FOR N = 1 TO 4000
30 NEXT N
90 PRINT: LIST
RUN
```

Observando el tiempo de ejecución de este programa, veremos que tardan los 4000 pases del bucle **FOR-NEXT** aproximadamente unos 10 segundos. A continuación haremos que N sea un entero, añadiendo la siguiente línea al programa:

```
10 DEFINT N
RUN
```

Comprobará que el tiempo se ha reducido a unos 7 segundos, dado al aumento de los 400 pases por segundo a 570 pases.



La función de conversión **CINT** (#) convierte un número a precisión entera. Entre este nuevo programa y ejecútelo:

```
NEW
1 CLS: PRINT
20 X = 12345.67890
30 PRINT X
60 PRINT CINT(X)
90 PRINT: LIST
RUN
```

Lo que aparece en pantalla es lo siguiente:

```
12345.67890
12345
```

La línea 30 imprime el número del valor X, que es el mismo que el de la línea 20

En la línea 60 se imprime el entero del valor X, dando el mismo resultado que cuando se usaba la sentencia **INT**.

Ahora cambiemos el valor X a negativo y veremos lo que pasa:

```
20 X = -12345.67890
RUN
```

Otra vez se consigue el mismo resultado que cuando se usaba la sentencia **INT**, la cual redondeaba la cifra a –12346.

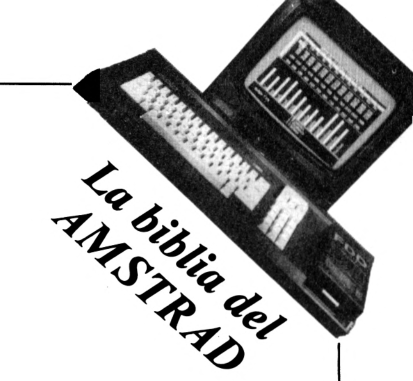
El Basic incluye un número de funciones matemáticas, las cuales son muy sencillas y fáciles de usar, pero si sus nociones de matemáticas no son muy buenas, deberá refrescar su memoria para poder entenderlas.

INT (N)

Hemos estudiado la función **INT** en detalle en otro Capítulo, así pues no cubriremos aquí este tipo de función.

FIS (N)

La función **FIX**, es muy parecida a la de **INT**, con la



única diferencia que en vez de redondear los números negativos hacia abajo, desestima todo lo que hay a la derecha del punto decimal. Vea en este simple ejemplo la diferencia:

```
PRINT INT(-12345.67)

produce -12346

PRINT FIX(-12345.67)

produce -12345

SQR(N)
```

Esta función lo único que produce es una raíz cuadrada. Entre el siguiente programa:

```
NEW
10 INPUT "LA RAIZ CUADRADA DE"; N
20 PRINT "ES "; SQR(N)
30 PRINT
40 GOTO 10
RUN
```

Otra forma de conseguir la raíz cuadrada de un número es usando el símbolo de la flecha hacia arriba (^), naturalmente significa «elevada a la potencia». Encontrar la raíz cuadrada de un número, es lo mismo que elevar a la potencia de 1/2. Cambie la línea 20 a:

```
20 PRINT "ES "; N[(1/2)
RUN
```

El problema está cuando queremos saber que la respuesta es correcta, con lo cual pondremos unas líneas al programa que harán que la raíz pase a la potencia de 21. Entre este programa para demostrarlo:

```
NEW
10 INPUT "LA RAIZ VENTIUNAVA DE"; N
15 R = N[(1/21)
20 PRINT "ES "; R
```




```
30 PRINT
33 PRINT R; "A LA POTENCIA DE 21 = "; R[21
36 PRINT
40 GOTO 10
RUN
```

EJERCICIO 28-1

Pitágoras descubrió que los lados de un triángulo rectángulo obedecen a la siguiente fórmula: C[2 = A[2 + B[2, donde C es el lado más largo (hipotenusa). Otra forma de expresar esta fórmula es que la longitud del lado C es igual a la raíz cuadrada de la suma de los lados rectos A y B. Si A = 5 y B = 12, escriba un programa para calcular la longitud del lado C

ABS (N)

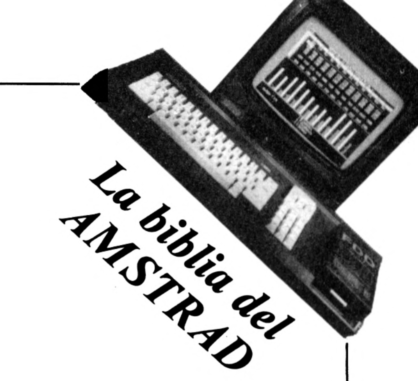
El valor absoluto tiene que ver mucho con los signos o sin ellos, dado que el valor absoluto de cualquier número, es un número sin signo. Si Vd. lo ha olvidado, este programa la refrescará la memoria.

```
NEW
10 INPUT "ENTRAR UN NUMERO "; N
20 A = ABS(N)
30 PRINT A
40 PRINT
50 GOTO 10
RUN
```

LOG (N)

Un logaritmo **(LOG)** es un exponente de una base de un número. El sistema **LOG** está centrado alrededor de lo que se llama logaritmos naturales, que usa el número 2.718282 como base. Algunos Basic tienen una segunda opción de **LOG** usando 10 como base, como en nuestro sistema decimal.

Entrar el siguiente programa:




```
NEW
10 INPUT "ENTRE UN NUMERO POSITIVO"; N
20 L = LOG(N)
30 PRINT "EL LOGARITMO DE"; N ;"A LA BASE NATURAL ES ="; L
40 PRINT
50 GOTO 10
RUN
```

Entrar el número 100, y obtendrá una respuesta de 4.60517, lo que significa que 2.718282 es la potencia de 100 del número 4.60517.

EXP (N)

La función **EXP** también se la conoce como lo opuesto a **LOG**, y calcula el resultado de elevar el número a una potencia dada. Si se usan números mayores de 88, el Computador dará un **ERROR** de rebasamiento, así como si se usa **EXP** con números menores que –88.7, hace que se llegue al límite por defecto, y por lo tanto se producirá un valor cero.

Para demostrar esta función, entre el siguiente programa:

```
NEW
10 INPUT "ENTRE UN NUMERO"; N
20 A = EXP(N)
30 PRINT "2.718282 ES PUESTO A"; N ;"POTENCIA = "; A
40 PRINT
50 GOTO 10
RUN
```

Como prueba de su programa, entre el siguiente número:

```
4.60517
```

La base de un logaritmo natural puesto a esta potencia será igual a 100.

EJERCICIO 28-2

Escriba un programa donde las dos funciones (**LOG y EXP**) en orden opuesto, usando valores de N positivos y negativos (**LOG (EXP) (N) = N**).



CAPITULO 29

FUNCIONES TRIGONOMÉTRICAS

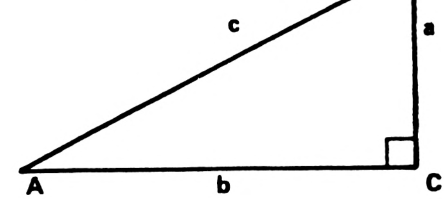
En este Capítulo no vamos a entrar de lleno en materia Trigonométrica, sino que solamente trataremos las funciones de **SIN, COS, TAN,** y **ATN**, para el cálculo de los ángulos de un triángulo por medio de un Computador.

En el triángulo que tenemos, el seno (**SIN**) del lado A es igual al lado a dividido entre el lado c y el coseno (**COS**) es igual al lado b dividido entre el lado c, así como la tangente es también igual al lado a dividido entre el lado b.

```
SIN A = a/c

COS A = b/c

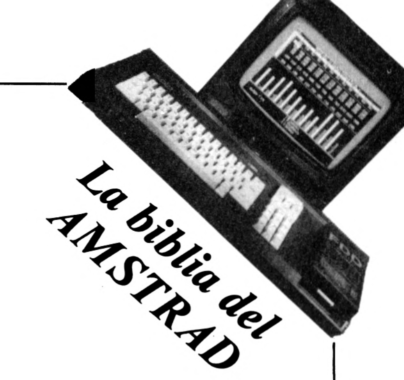
TAN A = a/b
```



Tomando esta relación podemos encontrar la proporción si sabemos el ángulo correspondiente, y para eso, escribiremos un programa:

124

(Continuará)




```
NEW
10 INPUT "ENTRE LOS GRADOS DE UN ANGULO ENTRE 0 Y 90"; A
20 S = SIN(A * 0.0174533)
30 PRINT "EL SENO DE UN ANGULO DE "; A ;" GRADOS ES "; S
40 PRINT: GOTO 10
RUN
```

Pruebe introducir los siguientes ángulos, 45, 30, 60, 90, 0, etc. Para la persona que no esté muy fuerte en trigonometría, encontrará rara la línea 20, y la razón es que los Computadores no miden los ángulos en grados, si no que su medida la hacen en radianes, siendo esta unidad igual a 57 grados aproximadamente, y la función de **SIN** no funcionará correctamente sin esta conversión.

Así pues para convertir grados a radianes, se deberá multiplicar los grados por 0.0174553

De la misma forma, para convertir radianes a grados, se deberá multiplicar los radianes por 57.29578

El coseno (**COS**) y la tangente (**TAN**) actúan de la misma forma, por lo que cambiaremos el programa residente de la siguiente manera:

```
NEW
10 INPUT "ENTRAR UN ANGULO ENTRE 0 Y 90 GRADOS"; A
20 T = TAN(A * 0.0174533)
30 PRINT "LA TANGENTE DE UN ANGULO DE "; A ;" GRADOS ES"; T
40 PRINT: GOTO 10
RUN
```

Así mismo el cálculo del coseno de un ángulo se podrá calcular cambiando las líneas 20 y 30:

```
20 C = COS(A * 0.0174533)
30 PRINT "EL COSENO DE UN ANGULO DE "; A ;" GRADOS ES"; C
RUN
```

El siguiente programa, imprimirá las tres funciones trigonométricas al mismo tiempo. Notará que en la línea 30 se divide la entrada entre 57.29578 en vez de multiplicarse por 0.0174533, siendo su resultado el mismo. Entre el siguiente programa:




```
NEW
10 CLS
20 INPUT "ENTRAR UN ANGULO ENTRE 0 Y 90 GRADOS"; A
30 A = A / 57.29578
40 PRINT
50 PRINT "ANGULO", "SENO", "COSENO", "TANG"
60 PRINT A * 57.29578, SIN(A), COS(A), TAN(A)
70 INPUT "DESEA HACER OTRO CALCULO (S/N)"; B$
80 IF B$ = "S" GOTO 20 THEN END
RUN
```

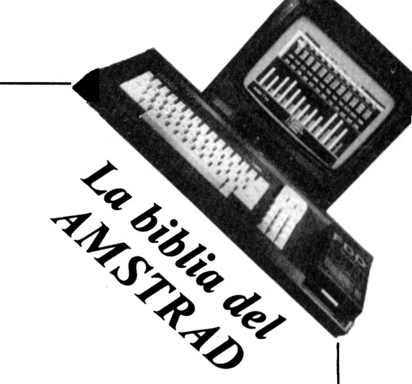
Funciones trigonométricas inversas

Lo opuesto para encontrar la relación entre dos lados de un triángulo cuando se conoce un ángulo, es calcular el ángulo cuando la relación entre los dos lados se conoce. En trigonometría, hay tres funciones que se conocen, pero los Computadores sólo admiten una, la cual se la denomina **ATN** (Arco de una Tangente).

En este programa, el Computador toma el ángulo entrado, imprime su tangente, tomando este resultado y lo computariza en arco de ángulo como comprobación. La letra I se usa en este programa dado que el arcotangente es el Inverso de la tangente. Entre el siguiente programa:

```
NEW
1 CLS
10 REM * DEMOSTRACION DE ATN *
20 INPUT "ENTRAR UN ANGULO ENTRE 0 Y 90 GRADOS"; A
30 T = TAN(A / 57.29578)
40 PRINT "TANGENTE = "; T,
50 I = ATN(T) * 57.29578
60 PRINT " ARCO DE LA TANGENTE = "; I
70 PRINT: GOTO 20
RUN
```

Si Vd. es una de esas personas que están familiarizados con la trigonometría conocerá las otras dos funciones inversas, **ARCSIN** y **ARCCOS**, pero dado que hay personas que se formarán un lío tremendo, usaremos las siguientes conversiones para crear este simple programa que si le interesa podrá salvarlo en cinta. Entre el siguiente programa:




```
NEW
10 CLS: REM * PROGRAMA DE DEMO DE FUNCIONES INVERSAS *
20 INPUT "ENTRE LA RELACION DE LOS DOS LADOS DE UN TRIANGULO"; R
30 AS = 2 * ATN(R/(1+SQR(ABS(1-R*R)))) * 57.29578
40 AC = 90 - AS : PRINT
50 PRINT "RELAC", "ARCSEN", "ARCCOS", "ARCTAN"
60 PRINT "(NUM)", "(GRAD)", "(GRAD)", "(GRAD)"
70 IF ABS(R) > 1 THEN 100
80 PRINT R, AS, AC, ATN(R) * 57.29578
90 PRINT: GOTO 20
100 PRINT R, "U", "U", ATN(R) * 57.29578
110 PRINT: GOTO 20
RUN
```

Recuerde, mientras que la Tangente puede ser cualquier número, cuando la relación se mueve fuera de –1 a 1, el Seno y Coseno no son definidos. Además, el **ARCTAN** y **ARCSIN** producen un ángulo entre –90 y 90 grados, pero el **ARCCOS** tiene una relación entre 0 y 180 grados.



CAPITULO 30

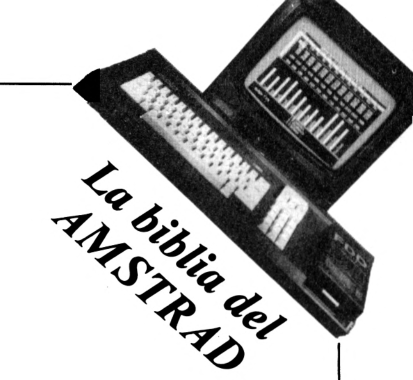
EL SONIDO

En este Capítulo cubriremos algunos aspectos de la generación de sonidos con este Computador, que además tiene un altavoz incorporado para reproducir dichos sonidos, así como se le puede conectar a un equipo de Alta Fidelidad para una reproducción mejor.

Empezaremos con un simple comando para generar un sonido que le podrá ser muy útil para llamar la atención en un programa donde le recuerde que el programa le está esperando. Introduciendo **PRINT CHR$ (7)**, el Computador producirá un tono de corta duración.

Entremos un programa para demostrar los sonidos el cual será examinado a continuación:

```
NEW
10 CLS: PRINT TAB(19)"MENU"
20 PRINT: PRINT TAB(3)"1. MUSICA"
30 PRINT TAB(3)"2. EXPLOSION"
40 PRINT TAB(3)"3. RUIDO DE AGUA"
50 PRINT TAB(3)"4. MOTORES"
60 PRINT: PRINT"SELECCIONE UN NUMERO"
70 GOSUB 1000
80 IF J<1 OR J>4 THEN PRINT "NUMERO INCORRECTO, PRUEBE OTRA VEZ": GOTO
10
90 ON J GOSUB 500,500,500,500
```




```
100 END
500 PRINT "EN ESTA POSICION PODRA INTRODUCIR SU PROGRAMA": PRINT
510 RETURN
1000 K$ = INKEY$: IF K$ <>"" THEN J = VAL (K$): RETURN
1010 PRINT CHR$(7);: FOR N = 1 TO 500: NEXT
1020 GOTO 1000
RUN
```

Una de las líneas más importantes de este programa es la línea 1000, la cual usa una función **INKEY$** que la trataremos de explicar en sucesivos Capítulos. Por consiguiente la línea 1000 detecta la tecla con **K$ = INKEY$**, y si es presionada **J = VAL (K$)** tomará el valor numérico para que se use en el resto del programa. La línea 1010 produce un corto sonido, usando **PRINT CHR$ (7)**, el cual tiene un punto y coma al final para prevenir que la pantalla salte.

Para producir sonidos, se deberá usar un comando simple o uno complicado, dependiendo del tipo de sonido que Vd. desee obtener. En este Basic, la instrucción para producir sonidos es la de **SOUND** que deberá ir seguida de por lo menos dos números, donde el primero será el canal seleccionado, existiendo tres canales en este Computador, y colocando cada nota a un canal seleccionado. La tabla de canales la mostramos a continuación:

No. seleccionado	Canal(es) selecionados
1	A
2	B
3	A y B
4	C
5	A y C
6	B y C
7	A y B y C

Como notará, los canales se les denomina como **A, B** y **C** pudiéndose seleccionar uno o más dependiendo del número seleccionado. Números mayores de siete, producirán efectos más complicados, los cuales los trataremos más adelante.



El segundo número que sigue en la instrucción **SOUND** corresponde al tono de la nota, controlando con una extensión desde 1 (tono más alto) al 4095 (tono más bajo), siendo la extensión más práctica la de 10 a 1000, dado que las notas por encima de 1000 producirán unos chasquidos en el altavoz. Las equivalencias musicales del número de notas se muestran claramente en su Manual en el Apéndice VII página 2.

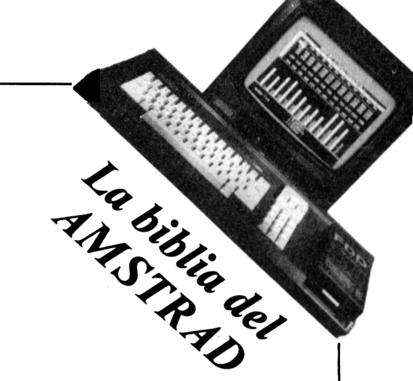
Empecemos nuestro estudio de la instrucción **SOUND** con una nota simple, introduciendo este programa:

```
NEW
10 SOUND 1,478,50
20 FOR N = 1 TO 2000: NEXT
30 SOUND 1,478,100
40 FOR N = 1 TO 2000: NEXT
50 SOUND 1,478,200
RUN
```

En la línea 10 se selecciona el canal A y el tono de la nota con el valor 478, además del tiempo, que corresponde a un quinto de segundo. Cuando el Computador ejecuta la instrucción **SOUND**, extrae los números y pasa a su sistema de sonido, que está separado del resto, por lo que el Computador seguirá ejecutando la siguiente línea que es un bucle de retardo, para poderse ejecutar la siguiente línea de sonido, y así sucesivamente, dado que si pusiéramos todas las instrucciones de **SOUND** seguidas el sistema las mezclaría sin producir la nota deseada.

El siguiente programa es una variación del primero mejorándolo entre notas:

```
NEW
10 SOUND 1,478
20 WHILE SQ(1)>128: WEND
30 FOR N = 1 TO 100: NEXT
40 SOUND 1, 478,50
50 WHILE SQ(1)>128: WEND
60 FOR N = 1 TO 100: NEXT
```




```
70 SOUND 1,478,200
RUN
```

La novedad en este programa está en la línea 20, donde **SQ** informa de la posición del sonido, igual que **EXPOS** informa la posición del cursor. **SQ** tiene un valor de por lo menos 128, que será comprobado por el bucle **WHILE... WEND** para hacer que el Computador espere a la línea de sonidos y así el siguiente comando no sea añadido hasta que el primero termine y conseguir una separación entre notas.

El siguiente paso será el hacer un programa donde se reproduzca una escala musical, que en este caso será la de C Mayor.

```
NEW
10 CLS: PRINT TAB(13)"ESCALA DE C MAYOR": PRINT: PRINT
20 FOR N = 1 TO 8
30 READ nota
40 SOUND 2,nota,100
50 NEXT
60 PRINT "Ahora todas las notas estan en fila"
100 DATA 478,426,379,358,319,284,253,239
RUN
```

A continuación pondremos una serie de programas para que Vd. practique y haga cambios, y así ver sus efectos, siempre ayudado de las tablas de su manual de usuario.

El primer programa será una armonía usando los tres canales.

```
NEW
10 FOR N= 1 TO 9
20 READ C1,C2,C3,P
30 SOUND 1,C1,P,7
40 SOUND 2,C2,P,5
50 SOUND 4,C3,P,5
60 NEXT
70 END
500 DATA 478,319,956,100
510 DATA 506,319,851,70
```




```
520 DATA 478,319,956,210
530 DATA 426,319,851,70
540 DATA 379,253,758,140
550 DATA 319,268,638,70
560 DATA 358,284,716,70
570 DATA 379,319,758,100
580 DATA 426,319,851,140
RUN
```

El siguiente programa demostrará cómo un generador de ruidos puede producir sonidos de efectos especiales.

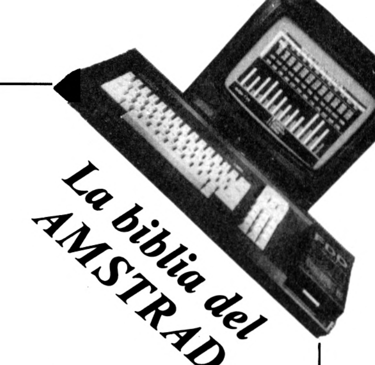
```
NEW
10 CLS: PRINT "OLAS EN LA PLAYA"
20 FOR N = 1 TO 10
30 FOR J = 15 TO 1 STEP -1
40 SOUND 2,0,20,7,0,0,J
50 NEXT: NEXT
60 PRINT "MARTILLAR DE UN MARTILLO"
70 FOR N = 1 TO 10
80 FOR J = 15 TO 0 STEP -1
90 SOUND 2,0,5,J,0,0,1
100 NEXT: NEXT
RUN
```

La sincronización

Dado que nos estamos quedando sin espacio para la temática de los sonidos por Computador, pues se necesitaría todo un libro para cubrirla, llegamos al final de este Capítulo con la Sincronización de los sonidos.

La sincronización se consigue usando los números de canales seleccionados, donde en la tabla que ponemos a continuación se muestran en su totalidad.

Numero selec.	Efecto
8	Sincronizacion con el canal A
16	Sincronizacion con el canal B
32	Sincronizacion con el canal C
64	Mantenido
128	Limpia los buffers de sonido



Así como los números que someten a las señales de sonido a ir a los tres canales, hay tres números que causan la sincronización, uno que mantiene la nota esperando en la fila, otro que quita todo lo que hay en la fila y uno que obtiene rápidamente el silencio. Vd. puede conseguir más de un efecto a la vez, sumando todos los números.

El siguiente programa demuestra este efecto:

```
NEW
10 SOUND 33,478,300
20 FOR N = 1 TO 2000: NEXT
30 SOUND 12,379,300
RUN
```

La línea 10 es un comando de sonido, el cual en una forma ordinaria sonaría de una vez, pero haciendo el selector de canales igual a 33, consistirá en el 1 del canal A más el número 32 que es la sincronización con el canal C. Después del retardo, el siguiente comando de sonido sería para el canal C, pero como se usa el número de selección 12, esto causa que el número 4 del canal C más el 8 se realice la sincronización con el canal A.

Como podrá comprobar, esto es realmente fácil aunque parezca difícil, así pues practique un poco y le cogerá el gusto a los sonidos.



CAPITULO 31

LA FUNCION INKEY$

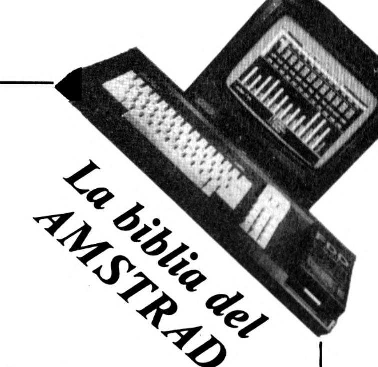
La potente función INKEY$ nos habilita la información de la sentencia INPUT del teclado sin tener que usar la tecla de **ENTER**.

Ejecute el siguiente programa para demostrarlo:

```
NEW
1 CLS
10 IF INKEY$ = "T" THEN 30
20 GOTO 10
30 PRINT "VD. HA PRESIONADO LA TECLA 'T'"
40 GOTO 10
RUN
```

Cuando se ejecuta este programa, parece que el teclado se queda muerto hasta que se presiona la tecla «T». Una vez que la comprobación pasa en la línea 10, la ejecución se mueve a la línea 30, y se imprime un mensaje, volviendo a empezar el proceso.

La forma que esta función trabaja es muy interesante, dado que el Computador busca una y otra vez la tecla que



se ha pulsado, y cada vez que se pulsa un tecla, ese carácter es almacenado en un área de su buffer, pudiendo tomar un carácter a la vez, o sea que cuando una tecla se pulsa, ese carácter reemplaza al anterior.

Si deseamos que la comprobación cubra más de un carácter, dado que esta función sólo comprueba uno sólo cada vez, se deberá escribir un programa para que realice este tipo de función. Añadir estas líneas al programa para ver su efecto:

```
15 IF IKEY$ = "P" THEN 50
50 PRINT "VD. HA PRESIONADO LA TECLA 'P'": GOTO 10
RUN
```

Como Vd. puede comprobar, no tenemos ninguna respuesta cada vez que una tecla se presiona; esto es dado porque el buffer del **INKEY$** es reseteado a una cadena nula cada vez que esta función se usa. Liste (**LIST**) el programa para ver cómo se limpia el buffer cada vez que se usa el **INKEY$**.

Supongamos que se presiona la tecla T cuando la línea 15 empieza su ejecución, esta letra irá al buffer y esperará hasta que otra tecla sea presionada. Como la línea 15 continua su ejecución, la actual letra que está en el buffer (T) es comparada con la que se presiona (P), y dado que las dos no son iguales, el control pasa a la siguiente línea y vuelve a la línea 10. En esta línea se llama a la función **INKEY$**, y cuando comprueba su buffer, la letra T no está dado que se ha limpiado. En el caso de que queramos preservar el valor de T, deberemos almacenarla en una cadena variable temporal. Cambie las líneas 10 y 15 para ver su efecto:

```
10 A$ = INKEY$: IF A$ = "T" THEN 30
15 IF A$ = "P" THEN 50
RUN
```

Ahora conseguimos algo, dado que si igualamos la función **INKEY$** a una cadena variable, podremos almacenar este valor por el tiempo que nosotros queramos.



Otra forma de usar esta función, es si el buffer no encuentra la tecla presionada, se dice que lee una cadena nula, y esta cadena se representa con dos comillas (" "). El código **ASCII** para nulo es 0.

Para ver qué rápido la función **INKEY$** hace la búsqueda, probaremos este nuevo programa:

```
NEW
10 K$ = INKEY$: IF K$ = "" PRINT "NO SE HA ENTRADO NADA"
20 PRINT ,,, K$: GOTO 10
RUN
```

Entre cualquier carácter y verá lo que pasa con esta función.

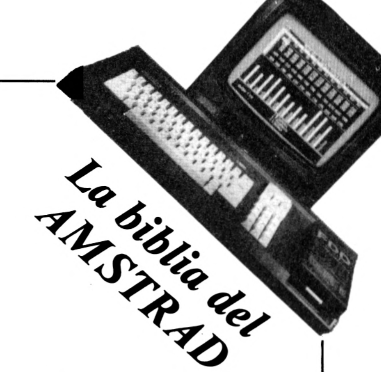
La idea general de cómo usar la función **INKEY$** es grandiosa, y solamente la experiencia y el uso continuado de ella, le podrá dar muchas satisfacciones, dado que sus prestaciones son muy prácticas.

El siguiente paso es demostrarle con un programa más serio, lo que se puede hacer con esta función. Así que escribiremos un programa donde Vd. tendrá que poner una clave secreta para poder abrir una puerta electrónica y así poder pasar a unas dependencias. La clave secreta es 1930. Y no deberá olvidarse el poner el punto al final de la cifra.

```
NEW
10 CLS: PRINT "ENTRE LA COMBINACION"
20 PRINT "SEGUIDO POR UN PUNTO"
30 PRINT "Y LA PUERTA SE ABRIRA"
40 K$ = INKEY$: IF K$ = "" GOTO 40
50 READ D$: IF D$ = "." GOTO 100
60 IF D$ = K$ GOTO 40
70 RESTORE: GOTO 40
100 CLS: PRINT "VD. PUEDE ENTRAR, ESPERE QUE SE ABRA LA PUERTA"
110 FOR T = 1 TO 2000: NEXT T: RESTORE: GOTO 10
1000 DATA 1,9,3,0,.
RUN
```

Una línea importante es la 40 que comprueba el buffer de la función **INKEY$** buscando si hay algo más que una cadena nula, y si encuentra un carácter, la ejecución pasa a la línea 50.

La línea 50 lee (**READ**) una parte de la **DATA**, y si en-



cuentra un punto, la ejecución pasa a la línea 100, pero si no lo encuentra, la ejecución pasa a la comprobación de la línea 60.

La línea 60 comprueba si el carácter entrado es igual al leído (**READ**) de la **DATA**, y si es correcto el pase, la ejecución vuelve a la línea 40 para esperar otro carácter del teclado, pero si es lo contrario que el carácter no concuerda con el existente en la **DATA**, la ejecución pasa a la línea 70.

En la línea 70, se restaura (**RESTORE**) el puntero de la **DATA** y vuelve al principio su ejecución, o sea a la línea 40 para empezar todo de nuevo.

Naturalmente Vd. podrá cambiar la palabra secreta a la que más le guste, con sólo modificar la línea de **DATA 1000**, y así demostrar a sus amigos lo inteligente que es su Computador.



CAPITULO 32

EL USO DEL PRINT USING

Las expresiones formateadas, se evalúan una a una sucesivamente, y se exponen a través de un cauce de acuerdo con la plantilla de formateo. Después de que se ha puesto cada expresión, se procesa la plantilla hasta encontrar una especificación adecuada de formato para dicho resultado. En el caso de no encontrarse el formato adecuado se producirá un tipo de **ERROR 5**.

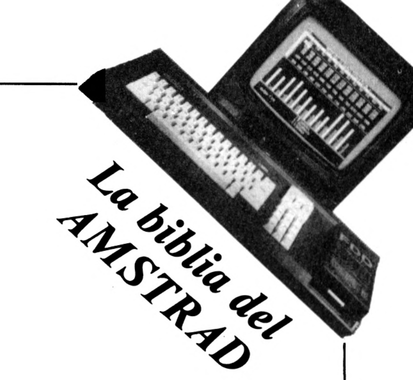
Hay que observar que el formato **USING** no efectúa separación de zonas de exposición ni comprobación de bordes de cauce.

Los caracteres de formateo de plantilla son los siguientes, y se describen más adelante:

! \ & # . + - * $ ^ , _

Cualquier otro carácter que se encuentre, el Basic lo sacará literalmente en pantalla.

A continuación pondremos un ejemplo de cómo usar el **PRINT USING** con números. Entre el siguiente programa:




```
NEW
10 A = 123.456789
40 U$ = "###.##"
50 PRINT USING U$; A
90 PRINT: LIST
RUN
```

La respuesta en pantalla será de:

```
123.46
```

La cifra ha sido redondeada en dos decimales. Añada las siguientes líneas al programa residente:

```
20 B = 1.6
60 PRINT USING U$; B
RUN
```

La respuesta en pantalla será:

```
123.46
  1.60
```

Lo primero que hay que notar, es que en el programa hemos llamado dos veces a la línea de formateo 40, una en la línea 50 y otra en la línea 60, y lo siguiente es que la dos cifras tienen los decimales alineados y en la segunda cifra se le ha añadido un cero para que se lea 1,60.

Pongamos más líneas a nuestro programa:

```
30 C = 9876.54321
70 PRINT USING U$; C
RUN
```

La respuesta será:

```
123.46
  1.60
%9876.54
```

La razón por la que aparece el signo %, es por que se ha sobrepasado la línea de formateo, dado que la capacidad de la misma es de tres dígitos antes del punto, y la tercera



cifra tiene cuatro. Cambiemos la línea 40 para arreglar esto:

```
40 U$ = "####.##"
RUN
```

El siguiente paso por el **PRINT USING**, será el uso del signo del Dolar ($), así pues haremos otro cambio en nuestro programa para ver sus resultados:

```
40 U$ = "$####.##"
RUN
```

El resultado será el siguiente:

```
$ 123.46
$   1.60
$9876.54
```

Pero supongamos que nosotros queremos que el signo del Dolar ($) aparezca inmediatamente al lado de la Cifra, tendremos que modificar la línea 40 para que se lea:

```
40 U$ = "$$####.##"
RUN
```

Entonces el resultado será:

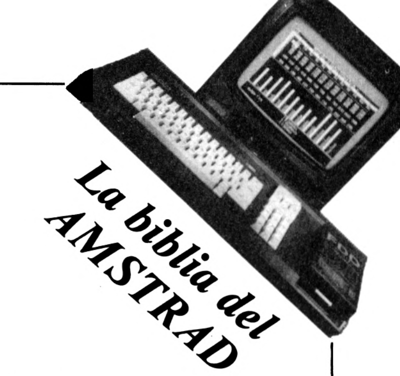
```
 $123.46
   $1.60
$9876.54
```

Hasta ahora hemos cubierto los siguientes puntos:

1.— El signo # con el **PRINT USIGN**, imprime todos los decimales en un mismo sitio por cada tamaño de número impreso.

2.— Redondea los decimales al número de signos # que se han puesto, pero no redondea las cifras a la izquierda del punto decimal, mandando una indicación de error (%) si la cifra sobrepasa el número de campos.

3.— Si se pone un signo de $ a la izquierda del campo de números, aparecerá alineado, lo mismo que aparece el



punto decimal. Este signo no aumenta el campo de caracteres.

4.— El uso de dos signos de Dólar ($$) a la izquierda del campo, hará que aparezca un signo $ inmediatamente al lado de la primera cifra de la izquierda. Uno de estos signos ($) puede reemplazar a un #, pero no aumentar su campo.

Otra forma de uso de esta función, es la impresión de cheques bancarios, donde a Vd. no le interesa que se pueda alterar la cifra en el mismo, por lo que se ha pensado el introducir una variante. Si nosotros cambiamos la línea 40 del programa residente:

```
40 U$ = "**###.##"
RUN
```

La respuesta en pantalla será la siguiente:

```
**123.46
****1.60
*9876.54
```

Esto significa que, poniendo dos asteriscos al principio del campo, cualquier espacio en blanco delante de la cifra será sustituido con asteriscos (*). Pero si además queremos que aparezca el signo $ delante, habrá que modificar la línea 40 otra vez (use el Editor y verá el tiempo que se ahorra):

```
40 U$ = "**$##.##"
RUN
```

Aparecerá en pantalla lo siguiente:

```
*$123.46
***$1.60
$9876.54
```

El siguiente paso, será el usar comas en el campo de caracteres, por lo que modificaremos de nuevo la famosa línea 40 para poder conseguir este efecto:




```
40 U$ = "**$,##.##"
RUN
```

En la pantalla aparecerá lo siguiente:

```
**$123.46
****$1.60
$9,876.54
```

Vemos que en nuestra pantalla sólo aparece una cifra con la coma, y eso quiere decir que la coma puesta antes de los campos numéricos sólo dividirán las cifras en grupos de tres.

Escribiremos de nuevo nuestro programa residente, para que vea las capacidades del **PRINT USING**:

```
NEW
1 CLS: PRINT
10 A = 123.456789
20 B = 1.60
30 C = 9876.54321
40 U$ = "#####.##      #####.##      #####.##"
50 PRINT USING U$; A, B, C
90 PRINT: LIST
RUN
```

La información que aparece en pantalla, verá que se diferencia en que en esta ocasión las cifras están alineadas y separadas.

Cadenas literales usando PRINT USING

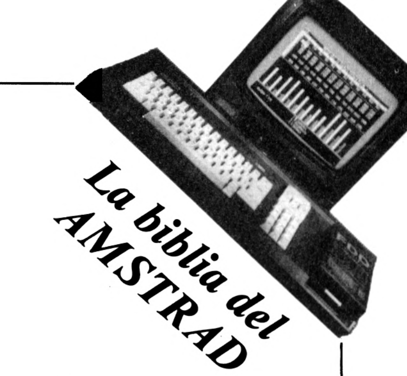
Los símbolos usados para las cadenas literales son los siguientes:

i Hará que sólo saque el primer carácter

\n espacios \Apárecerán los primeros n carácteres del dato literal, (habiendo por tanto n-2 espacios en blanco entre las barras inclinadas invertidas).

& Los datos literales se sacarán tal como son.

El signo más común que se usa en este formato es el de



las barras inclinadas invertidas (\), donde los espacios son equivalentes a los # cuando se usaban las cadenas numéricas, por lo que no entraremos de lleno en esta materia, y sí pondremos un programa donde se mezclen los dos tipos de cadenas.

Entre el siguiente programa, y sálvelo en cinta para podérselo demostrar a sus amistades.

```
NEW
1 REM * CAMBIO DE MONEDA INTERNACIONAL *
2 REM * CIFRAS AL MES DE JULIO DE 1985 *
5 MODE 2
10 CLS
80 RESTORE: PRINT
100 INPUT "CUANTAS PESETAS DESEA VD. CAMBIAR "; D
110 PRINT: PRINT TAB(18)"AL CAMBIO ACTUAL OBTENDRA": PRINT
400 READ A$,A,B$,B: IF A$ = "FIN" THEN 80
460 P$ = "\                   \    ########.##
########.##"
470 PRINT USING P$;A$;D/A;B$;D/B
800 C = C+1: IF C<11 GOTO 400
900 FOR T = 1 TO 500: NEXT T: C = 0: PRINT: GOTO 400
1000 DATA U.S.A. DOLAR, 173.84
1010 DATA CANADA DOLAR, 128.01
1020 DATA FRANCIA FRANCO, 18.79
1030 DATA INGLATERRA LIBRA, 228.01
1040 DATA IRLANDA LIBRA, 179.51
1050 DATA SUIZA FRANCO, 68.30
1060 DATA BELGICA FRANCO, 2.84
1070 DATA ALEMANIA MARCO, 57.23
1080 DATA ITALIA LIRA, .0897
1090 DATA HOLANDA FLORIN, 50.78
1100 DATA SUECIA CORONA, 19.87
1110 DATA DINAMARCA CORONA, 15.95
1120 DATA NORUEGA CORONA, 19.88
1130 DATA FINLANDIA MARCO, 27.57
1140 DATA AUSTRIA CHELIN, 8.1453
1150 DATA PORTUGAL ESCUDO, .998
1160 DATA JAPON YEN, .7012
1170 DATA AUSTRALIA DOLAR, 116.44
1500 DATA FIN, 0, FIN, 0
RUN
```

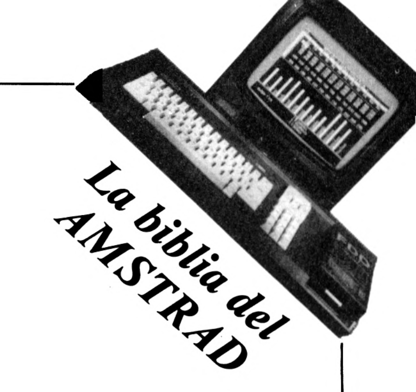
Como Vd. podrá observar en este programa, se mezclan tanto cadenas numéricas como cadenas literales, así pues es un buen ejemplo de cómo usar la función **PRINT USING**.



EJERCICIO 32-1

Escriba un programa tomando como base el anterior ejemplo, donde al ejecutarlo aparezca el siguiente resultado, naturalmente usando la función **PRINT USING**.

```
                        CREDITOS     IMPUESTOS       TOTAL
COMPUTADORES S.A.          18.3           .7          19.0
ELECTRICAS S.A.             1.8           .0           1.8
COMPONENTES S.A.            7.2           .3           7.5
                                     TOTALES          28.3
```



CAPITULO 33

LOS CONJUNTOS

Sabemos, que en este mundo de la informática, se pueden usar como variables cualquier combinación de las 26 letras del abecedario, y cualquier dígito del 0 al 9. Asimismo sabemos que no hay ningún programa que use el total de combinaciones, aunque algunos programas requerirán un gran número de ellas, usando como nombres diferentes tipos de datos para almacenarlos, y luego poderlos recuperar fácilmente.

La forma para resolver este problema es usando los **CONJUNTOS**, que son todos iguales con la única diferencia de su tamaño. Supongamos que tenemos placas de licencias del 1 al 10, y queremos usar el Computador para encontrar el tamaño del motor del vehículo a identificar. Esto no resultará nada difícil, pero antes de descubrir su potencial, aprenderemos paso a paso su desarrollo.

Supongamos que tenemos las siguientes licencias y tamaños de motores:



LICENCIAS	MOTORES
1	300
2	200
3	500
4	300
5	200
6	300
7	400
8	400
9	300
10	500

Ahora podremos dar a cada coche un nombre de letra distinto, usando las variables de la A a la J, pero esto sería una pérdida de tiempo y espacio de memoria, porque si usáramos mil coches, ¿que haríamos con ellos?

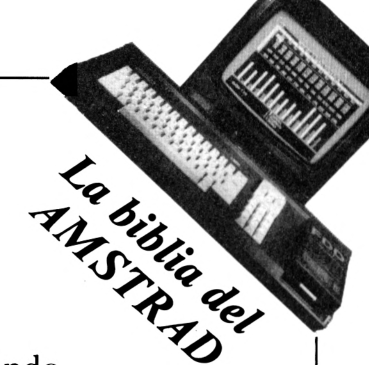
El Basic nos permite usar un nombre variable válido, como nombre de Conjunto, dado que el nombre del conjunto A no es lo mismo que la variable alfabética A y tampoco es lo mismo que la variable de una cadena A$. Así pues a nuestra variable la llamaremos A-sub, nombrando a los coches como A (1) al A (10).

Almacenaremos el tamaño de los motores de los coches en dos líneas de sentencias de DATA.

```
NEW
100 DATA 300,200,500,300,200
110 DATA 300,400,400,300,500
```

Notará que hemos tenido mucho cuidado en el almacenamiento de la numeración dentro de las sentencias de DATA, así el primer motor está en la primera sentencia de DATA, y la décima, en la décima localización.

Ahora deberemos poner el conjunto dentro de la memoria para direccionar inmediatamente estos elementos de datos. Piense lo difícil que es direccionar el séptimo motor, por ejemplo, usando lo que sabemos hasta el momento,



como las sentencias **DATA, READ** y **RESTORE** siendo muy enredado y lento.

La forma más fácil para crear un conjunto será de la siguiente forma:

```
50 FOR L = 1 TO 10
55 READ A(L)
60 NEXT L
RUN
```

No sale nada en pantalla!!!, pero eso no es cierto, dado que el Computador ha estado trabajando, y esto lo puede verificar viendo lo que tarda en salir la palabra Ready, y lo único es que no se imprimió lo que pasó.

Obviamente hemos usado un bucle **FORT-NEXT** para leer **(READ)** las 10 partes de **DATA** y nombrar los elementos que se han almacenado, A (1) al A (10). Usemos la sentencia **PRINT** para que se impriman los resultados.

```
200 FOR N = 1 TO 10
210 PRINT A(N)
220 NEXT N
RUN
```

Lo que hemos hecho, ha sido el leer los elementos de la **DATA** de un conjunto A (L), imprimiéndolos de un conjunto llamado A (N). La razón de esta diferencia es muy sencilla, al conjunto lo llamamos A y la localización de cada elemento es identificado dentro del conjunto por un número que lo hemos puesto entre paréntesis. Dentro de ese paréntesis, se podrá poner cualquier variable numérica y hasta una operación aritmética simple.

Trabajemos más en nuestro programa:

```
170 PRINT
180 PRINT "LICENCIAS", "MOTORES"
210 PRINT: PRINT N, A(N)
RUN
```

Ahora el programa una vez ejecutado tiene mejor aspec-



to. Hemos conseguido los números de licencias, números de motores, sin usar ninguna variable alfabética. Habiendo demostrado este punto, prosigamos, borrando las líneas 200, 210 y 220 de nuestro programa, y entremos las siguientes:

```
10 INPUT "QUE TAMANO DE MOTOR QUIERE VD. SABER"; W
210 PRINT W, A(W)
RUN
```

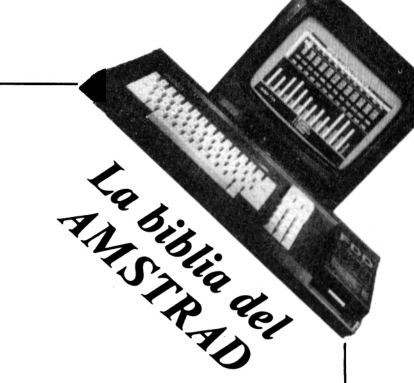
De esta forma se puede conseguir un simple programa de inventario para su negocio.

Vamos a ir un paso más en nuestro programa, y supongamos que hay cuatro colores (1, 2, 3 y 4) destinados a los diez coches, y así tendremos una lista como ésta:

LICENCIAS	MOTORES	COLORES
1	300	3
2	200	1
3	500	4
4	300	3
5	200	2
6	300	4
7	400	3
8	400	2
9	300	1
10	500	3

En el lenguaje profesional de Computadores, a esto se le llama una **MATRIZ**, la cual es un conjunto con más de una dimensión. Nuestro conjunto original tenía una dimensión horizontal de 2 y una vertical de 10, y por la misma regla, ésta tiene una dimensión horizontal de 3 y una vertical de 10.

Asignemos al conjunto destinado al código de colores una localizaciòn de 101 al 110, así como deberemos poner en el programa la información de los códigos de colores con la sentencia **DATA**, por lo que entraremos la siguiente línea a nuestro programa:




```
300 DATA 3,1,4,3,2,4,3,2,1,3
```

y

```
80 FOR S = 101 TO 110
85 READ A(S)
90 NEXT S
RUN
```

Podrá comprobar que los números de los elementos del 11 al 100 no se usan, así como del 111 al final de la memoria, dado que no se les ha asignado ningún valor.

Además verá que al ejecutar el programa le da un **ERROR** por falta de espacio, por lo que tendremos que dimensionar el conjunto A para que la memoria del Computador se la reserve y su ejecución sea correcta, por lo que introduciremos en nuestro programa una nueva función llamada **DIM**, y dado que el conjunto necesita 110, añadiremos la siguiente línea:

```
5 DIM A(110)
RUN
```

A continuación necesitaremos que la información salga correctamente en pantalla, por lo que cambiaremos las siguientes líneas:

```
10 INPUT "QUE NUMEROS DE MOTOR Y COLOR DESEA VD. PARA SU COCHE"; W
180 PRINT "LICENCIA", "MOTOR", "COLOR"
210 PRINT W, A(W), A(W+100)
RUN
```

Compruebe la respuesta de este programa con la anterior, cuando había una matriz de 2 dimensiones.

Si hasta ahora va entendiendo todo lo que se refiere a los conjuntos, Vd. podrá desarrollar su propio programa para satisfacer sus necesidades.

EJERCICIO 33-1

Supongamos que en el inventario de sus 10 coches, se le



incluye tres carrocerías diferentes, codificadas como 10, 20 y 30 y están en relación con la licencia del coche de la siguiente forma:

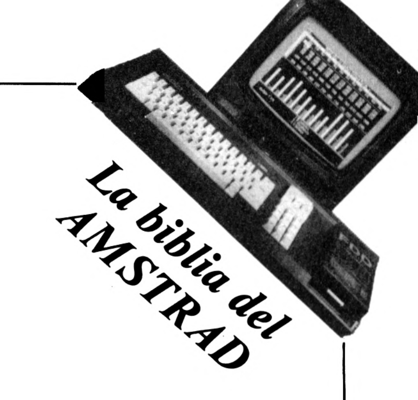
LICENCIA	CARROCERIA
1	20
2	20
3	10
4	20
5	30
6	20
7	30
8	10
9	20
10	20

Modifiquemos el programa residente para que se imprima el codigo de la **CARROCERIA** con el resto de la información, cuando el coche se identifica por su número de **LICENCIA**.

El dimensionamiento de un conjunto de cadena (alfabético), se hace lo mismo que es de un conjunto numérico, llamando a dicho conjunto como **A$ (DIM (A$ (N))**.

Los nombres que se le pueden dar a un conjunto son los siguientes:

```
A(N), BC(N), D3(N), E4$(N) y XY$(N)
```



CAPITULO 34

BÚSQUEDA (SEARCH) Y DISTRIBUCION (SORT)

Una de las especificaciones más potentes de un Computador, es la facilidad de búsqueda (**SEARCH**) en una información de **DATA** y distribuirla (**SPORT**) lo que se ha encontrado en algún orden, como alfabéticamente, inverso, numérico de más a menos, o a la inversa, etc.

Unas aplicaciones típicas de **SEARCH** y **SORT** son las siguientes:

1.— Colocación de una lista de clientes en orden alfabético.
2.— Distribución de una lista de correos por código postal.
3.— Distribución de una lista de clientes por código telefónico.

Empecemos con el programa. Tenemos ocho nombres diferentes, y queremos clasificarlos en orden albafético, así que los almacenaremos usando la función DATA:

```
NEW
1000 DATA BRAVO,XRAY,ALFA,ZULU,FOXTROT,TANGO,HOTEL,SIERRA
```



Dado que los clasificaremos por nombres en vez de por números, deberemos usar cadenas variables, conjuntos de cadenas, etc., por lo que también funcionará cuando se necesiten clasificar números.

Lo principal de una rutina **SORT** es un conjunto, así pues cada nombre tiene que ser leído (**READ**) de la **DATA** en un conjunto, por lo tanto escriba las siguientes líneas a nuestro programa:

```
10 REM * CLASIFICACION DE CADENAS *
20 CLS: FOR D = 1 TO 8: READ A$(N): N = N + 1: NEXT D
```

La línea 10 es solamente un título de lo que se va hacer.

La líneas 20 primero borra la pantalla, carga el conjunto, con la sentencia **READ** los ochos nombres en el almacenaje **A$ (1)** al **A$ (8)**. N es simplemente un contador que seguirá a través de todo el programa. En este caso N = 8 dado que sabemos el número de nombres, pero en el siguiente programa no sabremos dicho número, así que dejaremos la N de la forma normalmente usada.

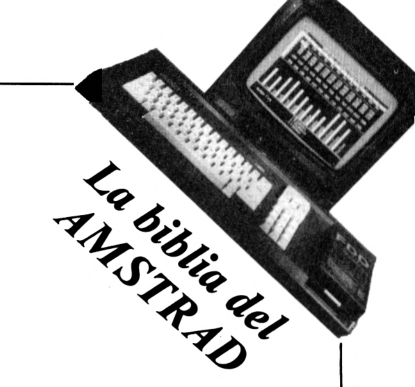
Lo importante en la rutina de **SORT** son los dos bucles **FORT-NEXT**, donde el primero, F, controlará el primer nombre, y, S, el segundo comparándolo con el primero.

Establezcamos nuestros bucles:

```
30 FOR F = 1 TO N-1     'F ES LA PRIMERA PALABRA A COMPARAR
40 FOR S = F+1 TO N     'S ES LA SEGUNDA PALABRA A COMPARAR
90 NEXT S               'HACE SIETE PASES
100 NEXT F              'HACE SIETE PASES
```

La **F** hace un bucle con los elementos del 1 al 7 (N–1=7)del conjunto **READ**, y **S** hacen un bucle con los elementos del 2 al 8 del conjunto **READ**, de esta forma nos da una diferencia de elementos de un conjunto para poderlos comparar.

A continuación saltaremos al final de nuestro programa y lo prepararemos para que salga impreso en pantalla.




```
110 FOR D = 1 TO N: PRINT A$(D),: NEXT D
```

Cuando la clasificación se ha terminado, los contenidos en **A$ (1)** al **A$ (8)** serán los mismos que se han leído de la **DATA**, pero clasificados en orden alfabético.

Para la rutina de **SORT** entre las siguientes líneas:

```
50 IF A$(F) <= A$(S) THEN 90 'COMPROBACION AL NUMERO MENOR ASCII
60 T$ = A$(F)                'ALMACENAMIENTO TEMPORAL DE LA 1 PALABRA
70 A$(F) = A$(S)             'COPIA DE LA 2 PALABRA AL PRIMER SITIO
80 A$(S) = T$                'PONER LA 1 PALABRA EN SEGUNDO SITIO
```

En la línea 50 se dice que si la primera palabra es más pequeña o igual a la segunda, se deje como está y se pase a la línea 90 donde terminará este pase y leerá (**READ**) otra palabra para compararla con **F**, y si no pasará a la siguiente línea.

En la línea 60 indica que se almacene la primera palabra en una celdilla temporal llamada **T$** y la mantendrá en ese sitio para un uso futuro.

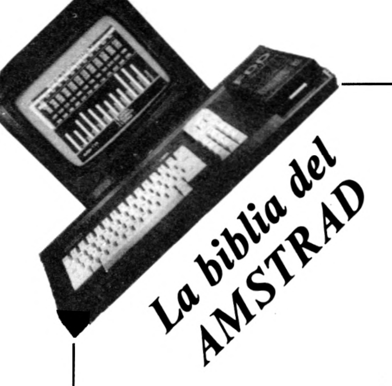
La línea 70 copia el nombre que tenemos en la segunda celdilla y la pone en la primera del conjunto. Si la segunda tiene una letra que va antes que la primera no se producirá nada.

La línea 80 completa el cambio, copiando el nombre que tenemos almacenado temporalmente en **T$**, dentro de la segunda celdilla del conjunto, así pues los contenidos en **A$ (1)** y **A$ (2)** han sido intercambiados con la ayuda de **T$**.

Si todo está bien, ejecute el programa con el comando **RUN**, y lo que aparecerá en pantalla será lo siguiente clasificado en orden alfabético:

```
ALFA     BARVO    FOXTROT   HOTEL
SIERRA   TANGO    XRAY      ZULU
```

Para poder ver con claridad la ejecución de este programa, será necesario el desacelerar su ejecución, e insertar unas líneas de **PRINT.**



Añada estas líneas al programa:

```
45 PRINT F; A$(F),, S; A$(S)
47 FOR Z = 1 TO 1000: NEXT Z
55 PRINT"    C A M B I A N D O"
85 PRINT F; A$(F),, S; A$(S)
RUN
```

Si no va lo suficientemente despacio, cambie la cifra de 1000 en la línea 47 por otra de más valor. Esto lo que pretende es que Vd. sea el Computador y haga las decisiones que la línea 50 debería hacer.

La explicación es la siguiente: en la celdilla #1 está la palabra **BRAVO**, y en la celdilla #2 está la palabra **XRAY** (como viene de la línea de **DATA**), de estas dos palabras, la de **BRAVO** es más pequeña (en número **ASCII**), así que la dejaremos en el primer sitio. Vayamos al siguiente pase de S. **BRAVO** está en la #1 y ALFA en la #3 pero **ALFA** es más pequeña (# **ASCII**) que **BRAVO**, así que deberemos cambiarlas:

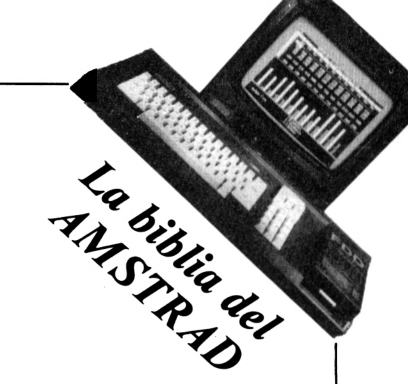
```
C A M B I A N D O
```

No se preocupe de lo que pasa en la segunda columna, dado que S está buscando en el conjunto y su contenido está cambiando y comparándolo con lo que hay en la primera columna, siendo lo que cuente lo que aparece en la primera columna que se incrementará alfabéticamente.

Mientras el programa sigue ejecutándose, mire las nuevas palabras que salen del segundo bucle y columna, y compárelas con la primera. Asimismo, tome nota del incremento de los números que salen a la izquierda.

Cuando sienta que tiene Vd. el programa dominado, añadiremos algo más a la impresión en pantalla, apareciendo lo que **T$** conserva mientras se realiza la clasificación. Ponga las siguientes líneas:

```
45 PRINT F; A$(F),, S; A$(S), "T$="; T$
85 PRINT F; A$(F),, S; A$(S), "T$="; T$
RUN
```



El siguiente paso, será el suministrar los datos para la línea de **DATA** desde el exterior (o sea desde el teclado), siendo muy similar el siguiente programa al primero y la lógica es idéntica. Cambie las siguientes líneas del programa residente:

```
5 D = 1
10 INPUT "SIGUIENTE NOMBRE"; A$(D): IF A$(D) = "FIN" GOTO 30
20 D = D + 1: N = N + 1: GOTO 10
borre la linea 1000 con DELETE
RUN
```

Entre diferentes nombres aleatorios, y cuando termine ponga la palabra FIN. La ejecución del programa en la pantalla será idéntica a lo que hemos visto anteriormente.

EJERCICIO 34-1

Cambie la línea 50 del programa residente, para que liste los nombres en orden alfabéticamente inversa.



CAPITULO 35

CONJUNTOS MULTIDIMENSIONALES

Hemos aprendido que los conjuntos no son nada más que áreas temporales de almacenamiento para números, caracteres alfabéticos, o para ambos, además de poder comparar los contenidos de las cadenas fuera de la matríz (o conjunto) con los que están dentro de ella.

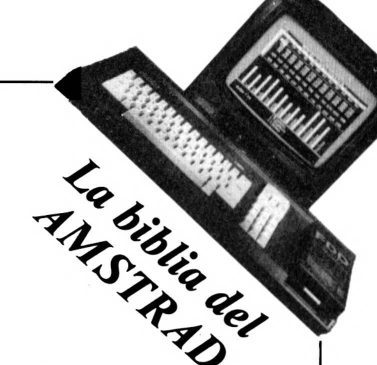
El conjunto donde solamente hay una dimensión, algunas veces se le llama Vector, pudiendo coger esa dimensión de conjunto y dividirla en cuatro partes iguales, posicionándolos uno al lado del otro, llamando a esto un conjunto de dos dimensiones, dado que están alineados en líneas y columnas.

Entrar el siguiente programa:

Nota: cualquier conjunto de más de 11 elementos debe ser dimensionado.

```
NEW
10 DIM M(52)
20 FOR V = 1 TO 52
30 PRINT V
40 NEXT V
RUN
```


(Continuará)



La ejecución de este programa, nos muestra las 52 posiciones y sus números (direcciones) todas en una columna, y a eso se le podrá llamar un Vector. Lo que no nos ha mostrado es el contenido de esas celdillas de memoria, así que para resolver este problema, cambiaremos el programa de la siguiente forma:

```
30 PRINT V, M(V)
RUN
```

En cada celdilla se ha almacenado el número 0, dado que cada valor se ha inicializado a cero al comienzo, así pues hasta ahora sabemos cómo encontrar la dirección de cada celdilla de memoria y cómo saber su contenido.

A continuación dividiremos las 52 celdillas en cuatro conjuntos iguales y los alinearemos uno al lado del otro, o sea que el conjunto de dos dimensiones, tendrá líneas y columnas. Entremos este nuevo programa:

```
NEW
10 DIM M(13,4): '13 LINEAS POR 4 COLUMNAS
20 FOR R = 1 TO 13
30 FOR C = 1 TO 4
40 PRINT R; C,
50 NEXT C
60 NEXT R
RUN
```

La ejecución del programa nos mostrará las direcciones de las 52 celdillas en la pantalla, todas a la vez. Pero para saber lo que cada celdilla contiene de **DATA**, se deberá cambiar la línea 40:

```
40 PRINT M(R,C)
RUN
```

Como podrá ver, el resultado es el mismo que la vez anterior, cero, dado que el programa se ha inicializado de nuevo. Así pues como las celdillas han de ser llenadas de datos de algún valor, esto se podrá conseguir leyendo **DATA**, introduciéndola desde el teclado, o de datos exter-



nos, como el de una cinta Cassette. En este caso llenaremos nuestra matriz desde una línea de **DATA**, así pues añadir las siguientes líneas:

```
100 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
110 DATA 21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37
120 DATA 38,39,40,41,42,43,44,45,46,47,48,49,50,51,52
```

y esta línea para leer la **DATA** y colocarla en la matriz:

```
35 READ M(R,C)
RUN
```

Ahora podemos ver los datos colocados en la matriz y cada matriz posicionada en una dirección específica. Quedémonos en este modo de comando durante un par de minutos, para interrogar al programa sobre diferentes posiciones de matrices y ver lo que contienen. Así pues cuando aparezca el cuadradillo del cursor, escribir lo siguiente:

```
PRINT M(2,3)
```

y la respuesta será 7. Ejecute de nuevo el programa y cuando tenga la respuesta del cursor escriba lo siguiente:

```
PRINT M(11,4)
```

y la respuesta será 44. Pruebe con lo siguiente:

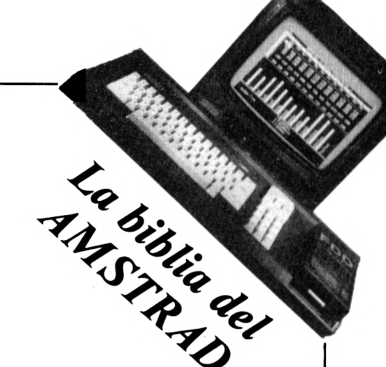
```
PRINT M(3,5)
```

Le dará un **ERROR** (Subscrip out of range), dado que en este programa no tenemos columna 5.

EJERCICIO 35-1

Cambie la línea 35 a una sentencia **LET** para llenar el conjunto con los mismo números, pero sin usar las líneas de **DATA**. (Asegúrese que cuando termine, ponga el programa como estaba originalmente.)

Hasta ahora hemos estado trabajando con números en



los conjuntos, pues bien, los cambiaremos para usar letras o palabras, con las mismas reglas que usamos anteriormente. Hagamos los suficientes cambios en nuestro programa residente (recuerde que después del ejercicio 35-1, debería haber puesto el programa en su estado original):

```
10 DIM M$(13,4)
35 READ M$(R,C)
40 PRINT M$(R,C)
RUN
```

El resultado no ha cambiado en nada, ahora tenemos una matriz de cadena, pero los datos son numéricos, por lo que tendremos que cambiar las líneas de **DATA**:

```
20 FOR R = 1 TO 6
100 DATA ALFA,BRAVO,CHARLY,DELTA,ECO,FOXTROT,GOLF,HOTEL
110 DATA INDIA,JULIA,KILO,LIMA,MIKE,NOVIEMBRE,OSCAR,PAPA
120 DATA QUEBEC,ROMEO,SIERRA,TANGO,UINFORM,VICTOR,WHISKEY
130 DATA XRAY,YANKEE,ZULU
RUN
```

Verá que no hay diferencia entre una matriz numérica y una de cadena, exceptuando que ésta maneja palabras.

El siguiente paso, será el mezclar los dos tipos de matrices, tanto numéricas como de cadenas alfabéticas.

El objetivo de este programa de demostración, es el poder llevar un cuaderno de notas de una Institución Benéfica de quién ha aportado y cuánto.

Empecemos este nuevo programa con las líneas de **DATA**:

```
NEW
1000 REM * FICHERO DE DATA *
1010 DATA 07.0182, PEREZ, 15
1020 DATA 07.0182, GONZALEZ, 87
1030 DATA 07.0182, RODRIGUEZ, 24
1040 DATA 07.0182, LOPEZ, 53
1050 DATA 07.0182, MARTINEZ, 42
```

Analizando las sentencias de **DATA**, empleamos dife-



rentes técnicas. El primer número de cada línea de **DATA** se llama compresión de datos, y significa que en un mismo número se representa más de una información, la cual contiene el mes, día y año en un número de 6 dígitos.

La segunda informaciòn, es que el primer número es protegido con un cero, dado que hasta octubre el mes está representado con solamente un número.

El segundo elemento de la información de la **DATA** es el nombre que aparece en cada línea. Se podría poner el nombre completo, pero si se usa una coma en medio, dicho nombre habría que ponerlo entre comillas.

El tercer elemento, representa a la cantidad percibida por Entidad.

Naturalmente estos datos son de demostración, y podrían almacenar cualquier otra que a Vd. se le ocurra.

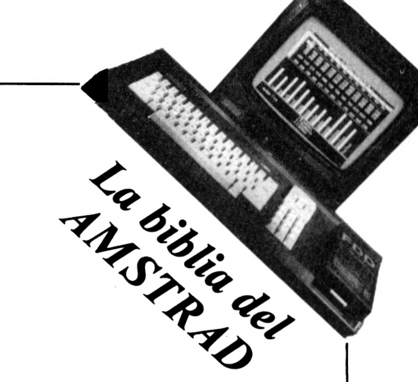
A continuación se deberá leer (**READ**) esta **DATA** en una matriz de cadena, por lo que añadiremos las siguientes líneas:

```
2 MODE 2
4 CLS
6 PRINT: PRINT "ENTRADA #","FECHA","NOMBRE","CANTIDAD"
10 FOR E = 1 TO 5: PRINT E,
20 FOR D = 1 TO 3
30 REM * ENTRAR LA DATA
40 READ R$(E,D)
50 PRINT R$(E,D),
60 NEXT D
70 NEXT E
RUN
```

Muy bien, la matriz se ha cargado y es confirmada en pantalla, viendo las 5 entradas con fecha del 1 de julio de 1982.

Ahora que sabemos que se carga bien, podemos quitar parte del Software.

Cambie la siguiente línea:




```
10 FOR E = 1 TO 5
borre con DELETE la linea 50
RUN
```

Muy bien, seguimos teniendo el encabezamiento pero el resto no aparece, así que ahora podemos interrogar a la matriz para sacar informes individuales.

Entre la siguiente línea:

```
5 INPUT "QUE INFORME NECESITA VD."; N$
RUN
```

Parece que funciona bien, así que contestaremos a la pregunta con un nombre que esté en una de las líneas de **DATA**, teniendo que hacer un rastreo de la matriz y comparar **N$** el nombre que hemos entrado con cada elemento, **R$ (E, D)**, hasta que se encuentre una semejanza, colocando el bucle **FOR-NEXT** otra vez y buscando cada elemento. Añada las siguientes líneas al programa:

```
110 FOR E = 1 TO 5
120 IF R$(E,2) = N$ THEN 150
130 NEXT E
140 PRINT " NO ESTA EN EL FICHERO": GOTO 90
150 PRINT E, R$(E,1), R$(E,2), R$(E,3)
160 PRINT: GOTO 90
RUN
```

Si Vd. tiene problemas en visualizar la línea 150, añada esta línea temporalmente, que imprimirá la dirección de cada elemento de **DATA** por debajo:

```
155 PRINT E, E;1, E;2, E;3
RUN
```

EJERCICIO 35-2

Escriba un programa donde se contenga en un conjunto de cadena de dos dimensiones con:




```
LOPEZ,C.         10439         100.00
PEREZ,I.         10023          87.24
RODRIGUEZ,J.     12936         398.34
GONZALEZ,F.      10422          23.17
```

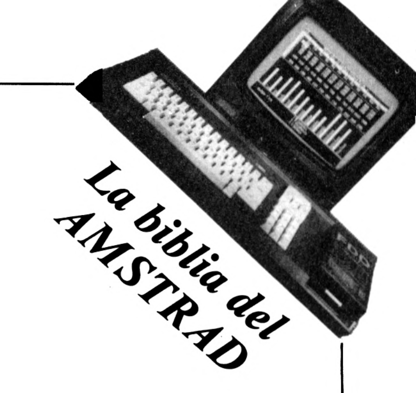
EJERCICIO 35-3

Clasifique los nombres del conjunto en el Ejercicio 35-2 alfabéticamente, sin olvidarse de mantener el resto de la información en cada línea del nombre original.

EJERCICIO 35-4

Con la información del ejercicio 35-2, pruebe a clasificar el conjunto en orden ascendente por el número de la columna 3.

APENDICES



APENDICE 1

Soluciones a los Ejercicios

Solución al Ejercicio 5-1:

```
50 PRINT D
```

Ejemplo de la ejecución del Ejercicio 5-1:

```
6000
```

Nota: Vd. probablemente habrá usado un número de línea diferente, pero lo importante es el usar la sentencia **PRINT**. Si no consiguió la respuesta adecuada, no se preocupe, teclee la línea 50 y ejecútela.

Solución al Ejercicio 5-2:

```
10 REM * SOLUCION AL TIEMPO SABIENDO LA DISTANCIA Y LA VELOCIDAD *
20 D = 6000
30 R = 500
40 T = D/R
50 PRINT "EL TIEMPO NECESARIO ES"; T ;"HORAS."
```

Ejemplo de la ejecución del Ejercicio 5-2:

```
EL TIEMPO NECESARIO ES 12 HORAS
```



Nota: Para llegar a la fórmula de la línea 40, es necesario partir de la fórmula original D = R * T, de donde T = D/R.

Solución al ejercicio 5-3:

```
10 REM * SOLUCION A LA CIRCUNFERENCIA *
20 P = 3.14
30 D = 35
40 C = P * D
50 PRINT "LA CIRCUNFERENCIA DEL CIRCULO ES"; C ;"METROS."
```

Ejemplo de la ejecución del ejercicio 5-3:

```
LA CIRCUNFERENCIA DEL CIRCULO ES 109.9 METROS.
```

Solución al Ejercicio 5-4:

```
10 REM * SOLUCION AL AREA DEL CIRCULO *
20 P = 3.14
30 R = 5
40 A = P * R * R
50 PRINT "EL AREA DEL CIRCULO ES"; A ;"METROS CUADRADOS."
```

Ejemplo de la ejecución del Ejercicio 5-4:

```
EL AREA DEL CIRCULO ES 78.5 METROS CUADRADOS.
```

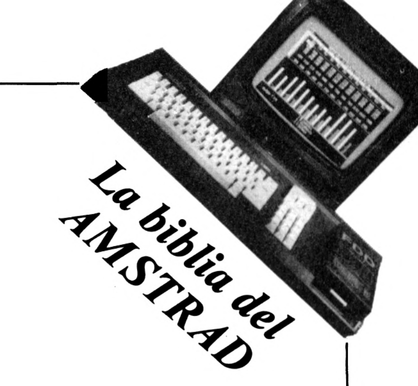
Solución al Ejercicio 5-5:

Teniendo en cuenta de que si llamamos a: C = cheques, D = depósitos, B = balance original, y N = nuevo balance.

```
10 B = 22500
20 C = 170 + 350 + 2250
30 D = 400 + 20000
40 N = B - C + D
50 PRINT "SU NUEVO BALANCE ES DE"; N ;"PESETAS"
```

Ejemplo de la ejecución del Ejercicio 5-5:

```
SU NUEVO BALANCE ES DE 40130 PESETAS
```



Solución al Ejercicio 6-1;

```
10 REM * SOLUCION AL TOTAL DE KILOMETROS RECORRIDOS *
20 N = 1000000
30 D = 10000
40 T = N * D
50 PRINT "EL TOTAL DE KILOMETROS CONDUCIDOS ES DE ";T
```

Ejemplo de la ejecución del Ejercicio 6-1:

```
EL TOTAL DE KILOMETROS CONDUCIDOS ES DE 1E+10
```

Nota: 1E + 10 es a equivalente un 1 seguido de 10 ceros (10.000.000.000)

Solución al Ejercicio 6-2:

```
20 N = 1E+6
30 D = 1E+4
```

Solución al ejercicio 7-1:

```
10 REM * CONVERSION GRADOS FAHRENHIET A CENTIGRADOS *
20 F = 65
30 C = (F - 32) * (5 / 9)
40 PRINT F; "GRADOS FAHRENHIET ="; C ;"GRADOS CENTIGRADOS"
```

Ejemplo de la ejecución del Ejercicio 7-1:

```
65 GRADOS FAHRENHIET = 18.3333 GRADOS CENTIGRADOS
```

Observe en donde se han puesto los paréntesis, y por regla general en caso de duda deberá usarlos.

Solución al Ejercicio 7-2:

```
30 C = F - 32 * (5 / 9)
```

Ejemplo de la ejecución del Ejercicio 7-2:

```
65 GRADOS FAHRENHIET = 47.2222 GRADOS CENTIGRADOS
```

Habrá notado que al Computador le ha costado bastan-

te el darnos la respuesta equivocada, dado de que le hemos dado una información errónea, el nos da la respuesta errónea también, eso demuestra que hay que programar al Computador correctamente.

Solución al Ejercicio 7-3:

```
30 C = (F - 32) * 5 / 9
```

Ejemplo de la ejecución del Ejercicio 7-3:

```
65 GRADOS FAHRENHIET = 18.3333 GRADOS CENTIGRADOS
```

Solución al Ejercicio 7-4:

Hay dos posibles respuestas:

```
30 - (9 - 8) - (7 - 6) = 28
30 - (9 - (8 - (7 - 6))) = 28
```

Ejemplos de programas:

```
10 A = 30 - (9 - (8 - (7 - 6)))
20 PRINT A
```

O modificando la línea 10:

```
10 A = 30 - (9 - 8) - (7 - 6)
```

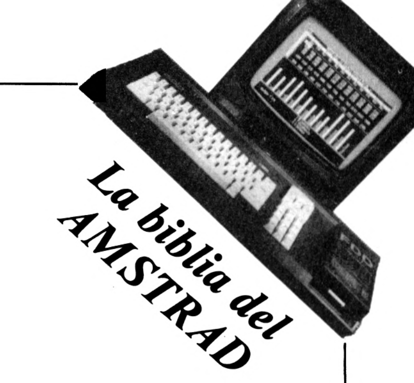
Pruebe algunas más que a Vd. se le ocurran.

Solución al ejercicio 8-1:

```
10 A = 5
20 IF A <> 5 THEN 50
30 PRINT "A ES IGUAL A 5"
35 PRINT
40 END
50 PRINT "A NO ES IGUAL A 5"
```

Ejemplo de la ejecución del ejercicio 8-1:

```
A ES IGUAL A 5
```



Solución al Ejercicio 8-2:

```
10 A = 6
20 IF A <> 5 THEN 50
30 PRINT "A ES IGUAL A 5"
40 END
50 PRINT "A NO ES IGUAL A 5"
60 IF A < 5 THEN 90
70 PRINT "A ES MAYOR QUE 5"
80 END
90 PRINT "A ES MENOR QUE 5"
```

Ejemplo de la ejecución del Ejercicio 8-2:

```
A NO ES IGUAL A 5
A ES MAYOR QUE 5
```

Nota: Hemos puesto una sentencia END en la línea 80 para que el programa no salte a la línea 90 después de haber imprimido la línea 70.

Solución al Ejercicio 13-1:

```
2 INPUT "CUANTOS SEGUNDOS DE RETARDO DESEA "; S
3 P = 400
4 D = S * P
5 FOR X = 1 TO D
6 NEXT X
7 PRINT "EL RETARDO HA TERMINADO, HA TARDADO"; S ;" SEGUNDOS"
```

Explicación:

La línea 2 usa una sentencia INPUT para obtener los segundos de retardo (S) deseados.

La línea 3 define a P como el número de pases necesarios para un retardo de un segundo.

En la línea 4 se multiplica el retardo para un segundo por el número de segundos deseados, definiendo al producto como D.

En la línea 5 da comienzo el bucle FOR-NEXT.

La línea 6 es la otra mitad del bucle.

La línea 7 informa que el retardo se ha terminado imprimiendo el número de segundos (S) que ha tardado.



Solución al Ejercicio 13-2:

```
60 PRINT "VELOCIDAD", "HORAS", "DISTANCIA"
65 PRINT "(Klm/H)", "(Tiempo)", "(Klm)"
```

Solución al Ejercicio 13-3:

```
1 MODE 2
5 CLS
10 PRINT          "******* T A B L A   DE   S U E L D O S ********"
20 PRINT
30 PRINT "ANUALIDAD", "MENSUAL", "SEMANAL", "DIARIO"
40 PRINT
50 FOR Y = 500000 TO 2500000 STEP 100000
55 REM * CONVERSION ANUAL EN MENSUAL *
60 M = Y / 12
65 REM * CONVERSION ANUAL EN SEMANAL *
70 W = Y / 52
75 REM * CONVERSION SEMANAL EN DIARIO *
80 D = W / 5
100 PRINT Y, M, W, D
110 NEXT Y
```

Ejemplo de la ejecución del ejercicio 13-3:

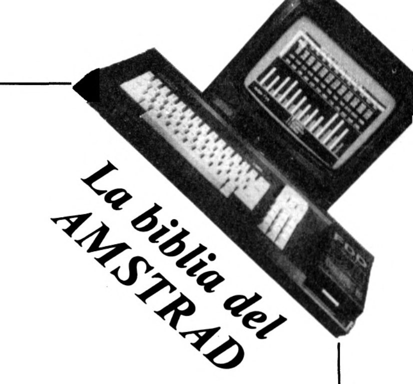
```
        *******  T A B L A   D E   S U E L D O S *******

ANUALIDAD          MENSUAL          SEMANAL          DIARIO

500000             41666.7          9615.39          1923.08
600000             50000            11538.5          2307.69
700000             58333.3          13461.5          2692.31
```

Solución al Ejercicio 13-4:

```
1 MODE 2
5 CLS
10 R = 100
20 D = 1
30 T = 100
40 PRINT "DIA", "DIARIO", "TOTAL"
50 PRINT " #", "PTS", "GANADO   "
60 PRINT
70 PRINT D, R, T
80 IF R > 1E+06 THEN END
90 R  = 2 * R
100 D = D + 1
110 T = T + R
120 GOTO 70
```



Ejemplo de la ejecución del Ejercicio 13-4:

```
DIA          DIARIO         TOTAL
 #           PTS            GANADO

1            100            100
2            200            300
3            400            700
4            800            1500
5            1600           3100
6            3200           6300
etc....
```

Solución del Ejercicio 13-5:

```
1 MODE 2
5 CLS
10 PRINT "ALAMBRE", "LARGO", "ANCHO", "SUPERFICIE"
20 PRINT "(METROS)", "(METROS)", "(METROS)", "(METROS CUADRADOS)"
25 PRINT
30 F = 1000
40 FOR L = 0 TO 500 STEP 50
50 W = (F - 2 * L) / 2
60 A = L * W
70 PRINT F, L, W, A
80 NEXT L
90 END
```

Ejemplo de la ejecución del Ejercicio 13-5:

```
ALAMBRE       LARGO        ANCHO        SUPERFICIE
(METROS)      (METROS)     (METROS)     (METROS CUADRADOS)

1000          0            500          0
1000          50           450          22500
1000          100          400          40000
1000          150          350          52500
1000          200          300          60000
1000          250          250          62500
1000          300          200          60000
etc......
```

Otra solución al Ejercicio 13-5:

En este programa el Computador se encargará de la comparación.

```
1 MODE 2
5 CLS
9 REM * PONER EL AREA MAXIMA A CERO *
10 M = 0
14 REM * PONER LA LONGITUD DESEADA A CERO *
15 N = 0
```




```
19 REM * F ES EL TOTAL DE LOS METROS DE LA PARCELA *
20 F = 1000
24 REM * L ES LA LONGITUD DE UN LADO DEL RECTANGULO *
25 FOR L = 0 TO 500 STEP 50
29 REM * L ES EL ANCHO DE UN LADO DEL RECTANGULO *
30 W = (F - 2 * L) / 2
35 A = W * L
39 REM * COMPARACION *
40 IF A <= M THEN GOTO 55
45 M = A
50 N = L
55 NEXT L
60 PRINT "ESTAS DIMENSIONES SON LAS MAXIMAS PARA:"
65 PRINT N; "MT. POR"; 500-N; "MT. PARA  UN AREA TOTAL DE" ; M ;  "MT.
CUADRADOS"
```

Ejemplo de esta ejecución:

```
ESTAS DIMENSIONES SON LAS MAXIMAS PARA:
250 MT. POR 250 MT. PARA UN AREA TOTAL DE 62500 MT. CUADRADOS
```

Solución del ejercicio 13-6:

```
5 MODE 2
10 REM * ENCONTRAR LA CARGA OPTIMA *
20 CLS
30 PRINT "CARGA", "CIRCUITO", "ORIGEN", "CARGA"
40 PRINT "RESIST", "POTENC", "POTENC", "POTENC"
50 PRINT "(OHMS)", "(VATIOS)", "(VATIOS)", "(VATIOS)"
60 PRINT
70 FOR R = 1 TO 20
80 I = 120 / (10 + R)
90 C = I * I * (10 + R)
100 S = I * I * 10
110 L = I * I * R
120 PRINT R, C, S, L
130 NEXT R
```

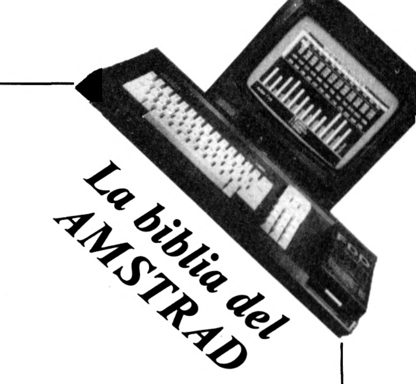
Ejemplo de la ejecución del Ejercicio 13-6:

```
CARGA           CIRCUITO        ORIGEN          CARGA
RESIST          POTENC          POTENC          POTENC
(OHMS)          (VATIOS)        (VATIOS)        (VATIOS)

1               1309.09         1190.08         119.008
2               1200            1000            200
3               1107.69         852.071         255.621
etc......
```

Solución del Ejercicio 14-1:

```
10 PRINT "EL ", "TOTAL ", "GASTADO"
20 PRINT "PRESUPUESTO              MES              ES"
30 PRINT TAB(0); "CATEGORIA"; TAB(16); "PRECIO"; TAB(32); "UNIDAD"
```



Solución al Ejercicio 14-2:

```
1 MODE 2
30 PRINT TAB(1); "ANUALIDAD"; TAB(12); "MENSUAL"; TAB(25); "SEMANAL";
35 PRINT TAB(38); "DIARIO"; TAB(51); "HORA"
85 REM *CONVERSION SEMANAL A HORA *
90 H = W / 40
100 PRINT TAB(0); Y;TAB(11); M;TAB(24); W;TAB(37); D;TAB(50); H
```

Ejemplo de la ejecución del ejercicio 14-2:

```
          *******  T A B L A   D E   S U E L D O S  *******

ANUALIDAD        MENSUAL          SEMANAL           DIARIO           HORA

500000           41666.7          9615.39           1923.08          240.385
600000           50000            11538.5           2307.69          288.462
etc..........
```

Solución del Ejercicio 14-3:

```
1 MODE 2
30 PRINT "INTERNA";TAB (10); "CARGA";TAB(21); "CIRCUITO";
35 PRINT TAB(36); "ORIGEN";TAB(51); "CARGA"
40 PRINT "RESIST";TAB(10); "RESIST";TAB(21); "POTENC";
45 PRINT TAB(36); "POTENC";TAB(51); "POTENC"
50 PRINT "(OHMS);TAB(10); "(OHMS)";TAB(21); "(VATIOS)";
55 PRINT TAB(36); "(VATIOS);TAB(51); "(VATIOS)"
120 PRINT "  10";TAB(11); R;TAB(20); C;TAB(35); S;TAB(50); L
```

Ejemplo de la ejecución del ejercicio 14-3:

```
INTERNA     CARGA      CIRCUITO      ORIGEN       CARGA
RESIST      RESIST     POTENC        POTENC       POTENC
(OHMS)      (OHMS)     (VATIOS)      (VATIOS)     (VATIOS)

  10          1        1309.09       1190.08      119.008
  10          2        1200          1000         200
  10          3        1107.69       852.071      255.621
etc......
```

Solución del Ejercicio 15-1:

```
10 FOR A = 1 TO 3
20 PRINT "BUCLE A"
30 FOR B = 1 TO 2
40 PRINT "  ", "BUCLE B"
42 FOR C = 1 TO 4
44 PRINT "  ", "  ", "BUCLE C"
48 NEXT C
50 NEXT B
60 NEXT A
```

173



Solución del ejercicio 15-2:

El programa de este Ejercicio es el mismo que el de la solución del 15-1, añadiendo las siguientes líneas:

```
45 FOR D = 1 TO 5
46 PRINT "   ", "   ", "   ", "BUCLE D"
47 NEXT D
```

Solución del Ejercicio 16-1:

Añadiendo esta línea al programa, le redondeará la cifra mejorando la lectura de la respuesta.

```
55 A = INT(A)
```

Solución del Ejercicio 16-2:

```
55 A = INT(10 * A) / 10
```

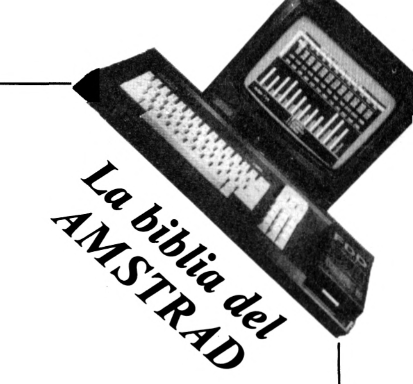
Cuando el valor 3.14159 es multiplicado por 10, se convierte en 31.4159. El entero de este valor será 31, y una vez dividido entre 10 se convierte en 3.1.

Solución del Ejercicio 16-3:

```
55 A = INT(100 * A) / 100
```

Solución del Ejercicio 17-1:

```
10 INPUT "ENTRE UN NUMERO CUALQUIERA"; X
20 T = SGN(X)
30 ON T + 2 GOTO 50, 60, 70
45 END
50 PRINT "EL NUMERO ES NEGATIVO."
55 END
60 PRINT "EL NUMÈRO ES CERO."
65 END
70 PRINT "EL NUMERO ES POSITIVO."
```



Solución del Ejercicio 22-1:

```
10 CLS
20 INPUT "PRIMERA CADENA"; A$
30 INPUT "SEGUNDA CADENA"; B$
40 PRINT: PRINT "ORDEN ALFABETICO:"
50 IF A$<B$ THEN PRINT A$, B$: END
60 PRINT B$, A$
```

Solución del Ejercicio 23-2:

```
10 CLS
20 INPUT "ENTRAR LA PALABRA SECRETA"; A$
30 FOR X = 1 TO 11
40 READ N
50 P$ = P$ + CHR$(N)
60 NEXT X
70 IF A$ = P$ THEN 100
80 PRINT "PALABRA INCORRECTA, LO SIENTO"
90 END
100 PRINT "PALABRA SECRETA CORRECTA, TIENE ACCESO"
110 DATA 65,66,82,69,32,83,69,83,65,77,79
```

Solución del Ejercicio 23-1:

```
10 CLS
20 INPUT "ENTRAR UNA CADENA"; A$
30 IF  LEN(A$)>10  THEN  PRINT  "SE  HA  EXCEDIDO  DEL  LIMITE  DE  10
CARACTERES."
```

Solución del ejercicio 24-1:

```
10 CLS
20 INPUT "ENTRAR SU CALLE Y NUMERO"; A$
30 A = VAL(A$)
40 PRINT: PRINT "EL NUMERO DE LA CALLE DE SU VECINO ES "; A + 4
50 PRINT: LIST
```

Solución del Ejercicio 24-2:

```
10 CLS
20 FOR X = 101 TO 120
30 A$ = STR$(X)
40 PRINT A$ + "WT",
50 NEXT X
60 PRINT: LIST
```



Ejemplo de la ejecución del Ejercicio 24-2:

```
101WT                 102WT                  103WT                104WT
105WT                 106WT                  107WT                108WT
109WT                 110WT                  111WT                112WT
113WT                 114WT                  115WT                116WT
117WT                 118WT                  119WT                120WT
```

Solución del Ejercicio 25-1:

```
10 CLS
20 INPUT "ESTE ES UN ORDENADOR INTELIGENTE"; A$
30 B$ = LEFT$(A$,1)
40 IF B$ = "S" THEN PRINT "AFIRMATIVO": END
50 IF B$ = "N" THEN PRINT "NEGATIVO": END
60 PRINT "ESTO NO ES UNA PREGUNTA DE SI O NO"
70 GOTO 20
```

Solución del Ejercicio 25-2:

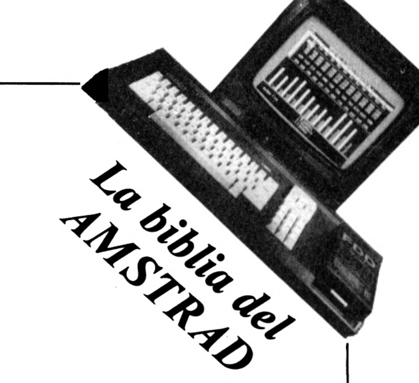
```
10 CLS: MAX$ = ""
20 FOR I = 1 TO 3
30 READ A$
40 N$ = MID$(A$,2,3)
50 IF N$ > MAX$ THEN MAX$ = N$: P$ = A$
60 NEXT I
70 PRINT "EL NUMERO DE PIEZA CON LA PORCION MAYOR NUMERICA ES"; P$
80 PRINT: LIST
90 DATA N106WT,A208FM,Z154DX
```

Solución del Ejercicio 28-1:

```
10 CLS
20 A = 5: B = 12
30 C = SQR(A[2 + B[2)
40 PRINT "LA RAIZ CUADRADA DE";A;"CUADRADO MAS";B;"CUADRADO ES";C
50 PRINT: LIST
```

Solución del Ejercicio 28-2:

```
10 INPUT "ENTRAR UN NUMERO"; N
20 PRINT "LOG( EXP(";N;")) ="; LOG(EXP(N))
30 PRINT "EXP( LOG(";N;")) ="; EXP(LOG(N))
40 PRINT
50 GOTO 10
```



Solución del Ejercicio 32-1:

```
5 MODE 2
10 CLS: PRINT TAB(24)"CREDITOS        IMPUESTOS       TOTAL"
20 FOR I = 1 TO 3
30 READ A$,X,Y,Z
35 REM    1234567890123456789012345678901234567890
40 U$ = "\                  \       ##.#      .#      ##.#"
50 PRINT USING U$; A$,X,Y,Z
60 NEXT I
70 READ A$,N
75 REM    1234567890
80 V$ = "\  \    ###.##"
90 PRINT TAB(35);: PRINT USING V$; A$,N
100 DATA COMPUTADORES S.A., 18.3, .7, 19.0
110 DATA ELECTRICAS S.A., 1.8, 0, 1.8
120 DATA COMPONENTES S.A., 7.2, .3, 7.5
130 DATA   "TOTALES", 28.3
```

Nota: las líneas 35 y 75 están como REM para facilitar la posición de los campos.

Solución del ejercicio 33-1:

Añada y cambie las siguientes líneas:

```
1 MODE 2
5 DIM A(210)
10 INPUT "QUE NUMERO DE MOTOR, COLOR Y CARROCERIA DESEA VD.  CONOCER";
W
130 FOR B = 201 TO 210
135 READ A(B)
140 NEXT B
180 PRINT "LICENCIAS", "MOTORES", "COLOR", "CARROCERIA"
210 PRINT W, A(W), A(W+100), A(W+200)
400 DATA 20,20,10,20,30,20,30,10,20,20
```

Solución del Ejercicio 34-1:

Cambie la línea 50 a:

```
50 IF A$(F) <= A$(S) THEN 90
```

Alterar el orden inverso de impresión:

```
110 FOR D = N TO 1 STEP -1: PRINT A$(D),: NEXT D
```

Solución del Ejercicio 35-1:

```
35 M(R,C) = 4 * (R-1) + C
```



Solución del Ejercicio 35-2:

```
10 CLS
20 FOR E = 1 TO 4
30 FOR D = 1 TO 3
40 REM * ENTRANDO DATOS *
50 READ R$(E,D)
60 PRINT R$(E,D),
70 NEXT D: PRINT
80 NEXT E: PRINT
1000 REM * FICHERO DE DATA *
1010 DATA "LOPEZ,C.", 10439, 100.00
1020 DATA "PEREZ,I.", 10023, 87.24
1030 DATA "RODRIGUEZ,J.", 12936, 398.34
1040 DATA "GONZALEZ,F.", 10422, 23.17
```

Solución del ejercicio 35.3:

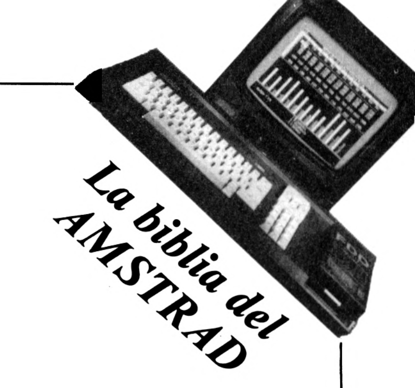
Añadir al programa:

```
100 REM * CLASIFICACION *
110 FOR F = 1 TO 3
120 FOR S = F + 1 TO 4
130 IF R$(F,1) <= R$(S,1) THEN 190
140 FOR J = 1 TO 3
150 T$(J) = R$(F,J)
160 R$(F,J) = R$(S,J)
170 R$(S,J) = T$(J)
180 NEXT J
190 NEXT S
200 NEXT F
210 PRINT: PRINT "CLASIFICACION ALFABETICA": PRINT
220 FOR E = 1 TO 4
230 FOR D = 1 TO 3
240 PRINT R$(E,D),
250 NEXT D: PRINT
260 NEXT E: PRINT
```

Solución del Ejercicio 35-4:

Cambie las siguientes líneas:

```
130 IF VAL(R$(F,3)) <= VAL(R$(S,3)) THEN 190
210 PRINT: PRINT "CLASIFICACION NUMERICA": PRINT
```



APENDICE 2

Algunos programas de interés

1.— Reloj de 24 Horas

```
10 INPUT"LA HORA ES"; E
20 F = INT(E/10): E = E - (F*10)
30 INPUT"LOS MINUTOS SON"; C
40 D = INT(C/10): C = C - (D*10)
50 INPUT"LOS SEGUNDOS SON"; A
60 CLS: PRINT CHR$(23)
70 B = INT(A/10): A = A - (B*10)
80 FOR N = 1 TO 400: NEXT N
90 A = A + 1
100 IF A > 9 THEN 120
110 GOTO 310
120 A = 0
130 B = B + 1
140 IF B > 5 THEN 160
150 GOTO 310
160 B = 0
170 C = C + 1
180 IF C > 9 THEN 200
190 GOTO 310
200 C = 0
210 D = D + 1
220 IF D > 5 THEN 240
230 GOTO 310
240 D = 0
250 E = E + 1
260 IF E > 9 THEN 280
270 GOTO 300
280 E = 0
290 F = F + 1
300 IF (F=1) AND (E=3) THEN A = 0: B = 0: C = 0: D = 0: E = 1: F = 0
310 LOCATE 20,6:PRINT F; E; ":"; D; C; ":"; B; A
320 GOTO 80
```



2.— COMPROBACION DE NUMEROS

Para los que tienen responsabilidades de inventarios, balances,etc., este programa le será de mucha utilidad para la comprobación de los diferentes códigos. Este programa está diseñado para números de seis cifras.

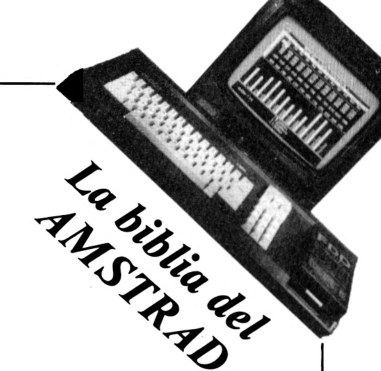
```
5 CLS
10 PRINT
20 INPUT "EL PRIMER DIGITO ES:"; A
30 INPUT "EL SEGUNDO DIGITO ES:"; B
40 INPUT "EL TERCER DIGITO ES:"; C
50 INPUT "EL CUARTO DIGITO ES:"; D
60 INPUT "EL QUINTO DIGITO ES:"; E
70 INPUT "EL SEXTO DIGITO ES:"; F
80 PRINT
90 PRINT "EL NUMERO ES:";A;B;C;D;E;F
100 S = A + 2 * B + C + 2 * D + E + 2 * F
110 T = INT(S/10)
120 U = S - T * 10
130 S = T + U
140 IF S > 9 THEN 110
150 PRINT "EL DIGITO COMPROBADO ES: "; S
```

3.— PROGRAMA PARA CALCULAR UNA ANTENA CUBICA QUAD

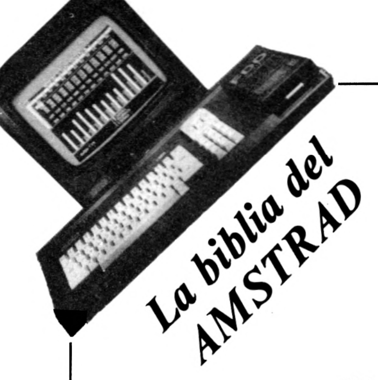
Las antenas del tipo cúbicas quad son de excelente ganancia, y se usan para las comunicaciones tanto en recepción como en transmisión. Eléctricamente, consta de dos lazos de cable, de los cuales a uno se le conecta un cable caxial, o paralelo, para obtener la señal, y el otro es un lazo completo.

Si desea que su resultado salga por impresora, lo único que deberá hacer, es sustituir todas las sentencias **PRINT** por **PRINT#8** y el cálculo de su antena lo tendrá en papel.

```
5 MODE 2
10 CLS
20 PRINT "DISENO DE SU PROPIA ANTENA DE ALTA GANANCIA""
30 PRINT
40 INPUT "FRECUENCIA CENTRAL EN MEGAHERCIOS = "; F
50 CLS
60 PRINT ">---> A N T E N A   C U B I C A   C U A D R A D A <---<
70 PRINT
80 E = .985 * F
90 G = 1.033 * F
100 D = 1000 / F
```




```
110 R = 1032 / F
120 B = 118 / F
130 X = (2 * (R * R / 64))
140 S = SQR(X)
150 X = (S * S + (B * B / 4))
160 P = SQR(X)
170 X = ((R * R / 64) + 75 * 75 / (F * F * 4))
180 T = SQR(X)
190 X = ((R * R / 64) + 125 * 125 / (F * F * 4))
200 U = SQR(X)
210 W = 468 / F
220 PRINT "LA FRECUENCIA  CENTRAL DE ESTA  ANTENA ES" ;  F ; "MHZ.  EN
ESTOS DOS"
230 PRINT  "ELEMENTOS SE  TENDRA UN  PORCENTAJE DE  ONDA REFLEJADA  DE
MENOS DE"
240 PRINT "2:1 SOBRE  UNA COBERTURA DE  FRECUENCIA DE ";  E ;"A"; G  ;
"MHZ"
250 PRINT "CUANDO SE USA UN CABLE DE 50 A 75 OHMS DE ALIMENTACION."
260 PRINT
270 PRINT "LA LONGITUD  DEL BRAZO CENTRAL  (BOOM) PUEDE VARIAR  ENTRE"
;75/F
280 PRINT "PIES Y" ;125/F; "PIES CON MUY POCO EFECTO. UNA LONGITUD DE"
;B; "PIES"
290 PRINT "SERIA LA MEJOR."
300 PRINT
310 PRINT "LA LONGITUD TOTAL DEL CABLE DEL ELEMENTO CONDUCTOR ES DE" ;
D ;"PIES,"
320 PRINT "QUE SERRA DE"; D/4 ;"PIES EN CADA UNO DE LOS LADOS."
340 GOSUB 1000
360 PRINT "LA LONGITUD TOTAL DEL  CABLE DEL ELEMENTO REFLECTOR ES  DE"
;R ;"PIES,"
370 PRINT "QUE ES IGUAL A"; R/4;" PIES DE CADA LADO."
380 PRINT
390 PRINT "LA LONGITUD MINIMA DE CADA BRAZO DE BAMBU, FIBRA DE  VIDRIO
O OTRO"
400 PRINT  "MATERIAL AISLANTE  FUERTE, SERA  DE "  ;S; "PIES,  MEDIDOS
DESDE EL "
410 PRINT "CENTRO DEL BRAZO (BOOM).  SI SE MONTA UNA ANTENA SIN  BRAZO
(BOOM),"
420 PRINT "CENTRAL, CADA BRAZO PARA MONTAR LOS CABLES DEBERA TENER POR
LO MENOS"
430 PRINT P; " PIES."
440 PRINT
450 PRINT  "EL RADIO  DE  MOVIMIENTO CIRCULAR  PARA DESPEJAR  LA  ZONA
ESTARA ENTRE"
460 PRINT T; " PIES  Y" ;U; " PIES, DEPENDIENDO  DE LA LONGITUD DE  SU
BRAZO CENTRAL"
500 GOSUB 1000: CLS
510 PRINT  "ESTA ANTENA  CUBICA, FUNCIONARA  PERFECTAMENTE EN  ALTURAS
BAJAS POR"
520 PRINT "ENCIMA DEL TERRENO, PERO SU RENDIMIENTO SERA MEJOR,  CUANDO
SE PONGA"
530 PRINT "A UNA ALTURA EQUIVALENTE A MEDIA LONGITUD DE ONDA ---"  ;W;
"PIES O MAS"
540 PRINT
550 PRINT "LA RELACION FRENTE-POSTERIOR  (LA REDUCCION DE LAS  SENALES
QUE NO SE"
560 PRINT "DESEAN RECIBIR)  EXCEDERA DE 10 DECIBELIOS  EN UN RANGO  DE
FRECUENCIA"
570 PRINT "DE" ; .97*F ; "A"  ; 1.03*F ; "MEGAHERCIOS, LLEGANDO A  LOS
25 DB EN"
580 PRINT "LA FRECUENCIA DE"; F ;"MEGAHERCIOS."
590 PRINT
600 PRINT "* * * B U E N O S   C O N T A C T O S * * *"
```




```
999 GOTO 999
1000 LOCATE 23,24:INPUT "PRESIONA <ENTER> PARA CONTINUAR"; A$
1010 CLS: RETURN
```

4.— TAMBORES

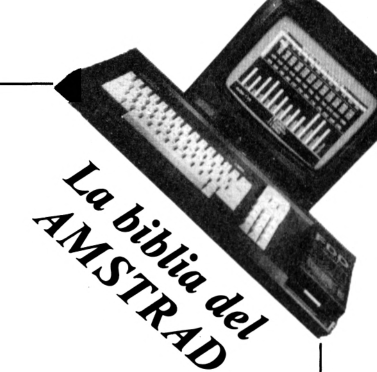
Este programa simplemente tocará un ritmo de tambores, consiguiendo con el comando **ENV** unos efectos muy realísticos.

```
10 CLS: S = 1
15 MODE 0
20 LOCATE 8,11: PRINT "TAMBORES"
30 ENV -1,1,-127,1,5,24,1
40 ENV 1,1,15,1,5,-3,2
50 ENV 2,15,-1,10
60 FOR T = 1 TO 4
65 IF S > 2 THEN S = 1
66 IF S = 1 THEN CH = 1
67 IF S = 2 THEN CH = 4
70 FOR Y = 1 TO 3
80 FOR O = 0 TO 100: NEXT O
90 TON = (T+4) * 54
100 SOUND CH,TON,6,0,1,1,1
110 NEXT
120 S = S + 1: NEXT: CH = 5: S = 1
130 FOR T O TO 90: NEXT
140 SOUND 2,0,150,15,2,0,1
150 FOR P = 1 TO 3
160 SOUND CH,486,6,15,1,1,2
170 FOR T = 0 TO 360: NEXT
180 SOUND CH,300,6,0,1,1,2
190 FOR T = 0 TO 180: NEXT
200 SOUND CH,300,6,0,1,1,2
210 FOR T = 0 TO 180: NEXT
220 SOUND CH,486,6,15,1,1,2
230 FOR T = 0 TO 360: NEXT
240 SOUND CH,300,6,0,1,1,2
250 IF P < 3 THEN FOR T = 0 TO 360: NEXT
260 IF P = 3 THEN FOR T = 0 TO 240: NEXT
270 NEXT P
280 GOTO 60
```

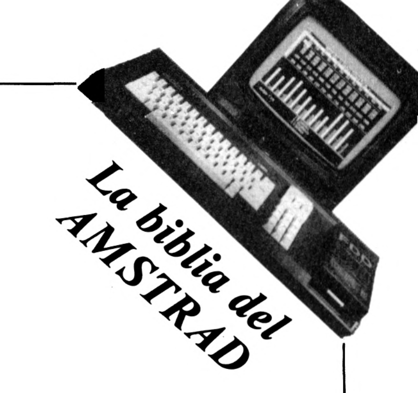
5.— COMANDO DE CATALOGO

Este programa de utilidad es de gran ayuda para poder analizar la dimensión de un programa, el cual lee la información de la cabecera del fichero del buffer de la Cassette.

```
180 ON BREAK GOSUB 260
190 R$ = CHR$(24); MODE 1
200 LOCATE 3,1: PRINT R$; " PONGA LA CASSETTE EN LA GRABADORA "; R$
```




```
210 LOCATE 2,6:PRINT "PRESIONAR";R$;" ESC ";R$;" ";R$;" ESC ";R$;
220 PRINT "CUANDO APAREZCA EL 'OK'"
230 PRINT:"PRINT
240 CAT
260 MODE 2
270 LOCATE 20,2:PRINT R$;" CATALOGO -- ANALISIS DE PROGRAMA ";R$
290 FIL$ = "": FOR I = 47244 TO 47259
300 FIL$ = FIL$ + CHR$(PEEK(I))
310 NEXT
330 BN = PEEK(47260)
350 BL = PEEK(47264) * 256 + PEEK(47263)
370 PL = PEEK(47269) * 256 + PEEK(47268)
390 LA = PEEK(47266) * 256 + PEEK(47265)
410 TYPE = PEEK(47262)
430 BLOCKS = PL / 2048
440 IF BLOCKS <> INT(BLOCKS) THEN BLOCKS = INT(BLOCKS) + 1
460 PLA = LA
470 IF BN > 1 THEN PLA = PLA - 2048: BN = BN - 1: GOTO 470
490 PE = PLA + PL
510 LOCATE 5,4:PRINT R$;" NOMBRE PROGAMA                         ";R$;
520 PRINT "      "; FIL$
530 LOCATE 5,6:PRINT R$;" TIPO DE PROGAMA                        ";R$;
540 PRINT "      ";
550 IF TYPE = 0 THEN PRINT"BASIC ESTANDAR"
560 IF TYPE = 1 THEN PRINT"BASIC PROTEGIDO"
570 IF TYPE = 2 THEN PRINT"CODIGO MAQUINA"
580 IF TYPE = 22 THEN PRINT"TEXTO ASCII"
590 LOCATE 5,8:PRINT R$;" NUMERO BLOQUE ACTUAL                   ";R$;
600 PRINT "      ";BN
610 LOCATE 5,10:PRINT R$;" BLOQUES TOTALES EN FICHERO            ";R$;
620 PRINT "      ";BLOCKS
630 LOCATE 5,12:PRINT R$;" BLOQUE DIRECCION DE CARGA             ";R$;
640 PRINT "      ";LA
650 LOCATE 5,14:PRINT R$;" LONGITUD DE BLOQUE                    ";R$;
660 PRINT "      ";BL;" BYTES"
670 LOCATE 5,16:PRINT R$;" DIRECCION DE CARGA DEL PROGRAMA       ";R$;
680 PRINT "      ";PLA
690 LOCATE 5,18:PRINT R$;" DIRECCION FINAL DEL PROGRAMA          ";R$;
700 PRINT "      ";PE
710 LOCATE 5,20:PRINT R$;" LONGITUD DEL PROGAMA                  ";R$;
720 PRINT "      ";PL:" BYTES"
730 PRINT "   ";STRING$(76,95)
750 LOCATE 10,23
760 PRINT "DESEA ANALIZAR OTRO PROGRAMA";
770 PRINT " (S/N) ";R$;"-";R$;CHR$(8);
780 K$ = INKEY$: IF K$ = "" THEN 780
790 IF K$ = "S" THEN RUN ELSE IF K$ = "N" THEN CLS: END
800 PRINT CHR$(7);: GOTO 750
```



APENDICE 3

Tabla de carácteres ASCII

DEC	OCTAL	HEX	ASCII	DEC	OCTAL	HEX	ASCII
0	000	00	NULO	64	100	40	@
1	001	01	SOH	65	101	41	A
2	002	02	STX	66	102	42	B
3	003	03	ETX	67	103	43	C
4	004	04	EOT	68	104	44	D
5	005	05	ENQ	69	105	45	E
6	006	06	ACK	70	106	46	F
7	007	07	BEL	71	107	47	G
8	010	08	BS	72	110	48	H
9	011	09	HT	73	111	49	I
10	012	0A	LF	74	112	4A	J
11	013	0B	VT	75	113	4B	K
12	014	0C	FF	76	114	4C	L
13	015	0D	CR	77	115	4D	M
14	016	0E	SO	78	116	4E	N
15	017	0F	OF	79	117	4F	O
16	020	10	DLE	80	120	50	P
17	021	11	DC1	81	121	51	Q
18	022	12	DC2	82	122	52	R
19	023	13	DC3	83	123	53	S
20	024	14	DC4	84	124	54	T
21	025	15	NAK	85	125	55	U
22	026	16	SYN	86	126	56	V
23	027	17	ETB	87	127	57	W
24	030	18	CAN	88	130	58	X
25	031	19	EM	89	131	59	Y
26	032	1A	SUB	90	132	5A	Z
27	033	1B	ESC	91	133	5B	[
28	034	1C	FS	92	134	5C	\
29	035	1D	GS	93	135	5D	]



30	036	1E	RS
31	037	1F	US
32	040	20	SP
33	041	21	!
34	042	22	"
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	-
46	056	2E	.
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?

94	136	5E	^
95	137	5F	_
96	140	60	`
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~

Este libro se acabó de imprimir en
Madrid. Gráficas Futura, Sdad.
Coop. Ltda., el 22-VII-86.

AMSTRAD CPC

OCR

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA



https://acpc.me/